UNIVERSITÉ GRENOBLE ALPES

ISBN: 978-2-11-129204-8

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Nano Électronique et Nano Technologies**

Arrêté ministériel : 7 août 2006

Présentée par

Saif Ur REHMAN

Thèse dirigée par **Mme. Lorena ANGHEL**

et codirigée par **M. Mounir BENABDENBI**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale d'Électronique, Électrotechnique, Automatique, et Traitement du Signal (EEATS)**

Développement des techniques de test et de diagnostic pour les FPGA hiérarchique de type mesh

Thèse soutenue publiquement le **06 Novembre 2015**,
devant le jury composé de :

M. Paolo PRINETTO
Professeur, Politecnico di Torino, Italy, Président
M. François PECHEUX
Professeur, UPMC, France, Rapporteur
M. Giorgio DI NATALE
Chargé de Recherche CNRS, LIRMM, France, Rapporteur
M. Mounir BENABDENBI
Professeur associé, Institut Polytechnique de Grenoble, France,
Co-directeur de thèse
Mme. Lorena ANGHEL
Professeur, Institut Polytechnique de Grenoble, France,
Directeur de thèse

Acknowledgments

First and foremost, I would like to thank my thesis director Prof. Lorena Anghel and co-director Prof. Mounir Benabdenbi for giving me the opportunity to be a part of the dynamic project collaborated with distinguished labs and an industrial partner. I am very thankful for their guidance and encouragement throughout the duration of my thesis. Their advice and support on conducting research and collaboration among project partners have been invaluable.

I would like to thank Prof. Paolo Prinetto for presiding over my thesis defense committee and also for his pertinent questions and valuable comments on my work. My special thanks to Prof. François Pêcheux and Dr. Giorgio Di Natale for reviewing my thesis manuscript and their constructive remarks. I value their great research experience and deep understanding of the challenges of the FPGA testing.

I would also like to express my gratitude to all my project partners. To Prof. Lirida Naviner from Telecom ParisTech, Prof. Habib Mehrez and Prof. Roselyne Chotin-Avot from UPMC Paris for their guidance and wonderful support. Their constructive remarks during project meetings and encouragement played a vital role in improving my work and steering it to the completion. My special thanks to Dr. Zied Marrakchi from Mentor Graphics for his immense support and guidance during implementation phases of my work. His suggestions and patience made my work easier and I have learned immensely from his expertise in the field of FPGA development. I would like to acknowledge the great help from Arwa Ben Dhia, Adrien Blanchardon and Emna Amouri throughout the project.

I would like to thank my colleagues and friends in TIMA for their help and support especially my dear friend Yi Gang whose brilliant suggestions provided me with the necessary relief during hard times. I also thank Muhammad Hamayun, Vladimir Pasca, Michael Dimopolous, Diarga Fall, Gilles Bizot, Hai Yu, Rshdee Alhakim and Mariam Abdallah for their wonderful company.

I must thank my dear friend Lihui Yang who has always been a source of support whenever I needed, and my old friend Abdullah Khan for his motivational talks and cheerful discussions.

Last, but certainly not the least, my sincere thanks to my parents Younus and Nasim and my sisters Nosheen, Misbah and Sobia for their prayers and encouragement. Their moral support made it easier for me to achieve my goals.

Abstract

Field Programmable Gate Arrays (FPGAs) are used in complex digital systems mainly due to their reconfigurability and shorter time-to-market. Maintaining high reliability of such systems in advanced technology nodes requires FPGAs to be highly robust as well as efficient in detecting faults if occur during life time of the chip. Therefore, FPGAs are aimed to be tested exhaustively for defects, making FPGA testing a challenging task. Efficient testing techniques require to perform FPGA test in all its modes of operation in least possible time. Among major DFT approaches, Built-In Self-Test (BIST) is considered as the most efficient technique for FPGA testing as it exploits very well the FPGA reconfigurability and its regular structure. FPGA cluster size as well as interconnect topologies exploration is an ongoing optimization process as it severely impacts the routability, area saving and testability of the FPGA. Multilevel interconnect in mesh of clusters FPGA is a novel approach that promises to give better routability and area saving compared to classical mesh FPGAs. Although, BIST is a generic technique, test configurations are architecture specific. Most of the existing BIST solutions target specific commercial FPGAs using their dedicated CAD tools, making it difficult to apply such solutions on a new FPGA architecture. In this thesis, we provide BIST schemes for a complete test and diagnosis of logic and interconnect resources in a novel hierarchical mesh FPGA. The proposed technique ensures full test and diagnosis by performing selection of test paths. It uses only 2x2 adjacent logic resources. Using this scheme, any NxN FPGA array can be further tested by N parallel 2x2 array procedure which ultimately reduces the test time. Another strategy for test time reduction based on joint testing of logic and intra-cluster interconnect is also proposed. In addition to this, a thorough analysis of the cluster size impact on the testability of given FPGA is performed. Moreover, BIST schemes are developed for the FPGA clusters enriched with different defect-tolerant techniques. BIST simulation results are produced for various cluster sizes as well as for different defect-tolerant FPGAs. Automated tools are developed to generate test configuration bitstreams and to integrate them into a standard FPGA CAD flow. Experimental results show that 100% test coverage for stuck-at and pair-wise bridging faults can be achieved with a multiplexer or even gate-level diagnostic resolution.

Key words: Built-In Self-Test, Hierarchical mesh of clusters FPGA, Multilevel interconnect, Off-line test and diagnosis, Logic and interconnect BIST.

Contents

Chapter 1 Introduction 14

1.1 Motivation 14

1.2 Key issues and contributions of the PhD thesis 18

1.3 Thesis organization 18

Chapter 2 Test and Diagnosis of FPGA: State of the art 20

2.1 Overview of the FPGA architecture 20

2.1.1 Classic mesh FPGA architecture 21

2.1.2 Variations in FPGA Interconnect 22

2.1.3 Multilevel mesh FPGA architecture 25

2.2 Fault modeling 27

2.2.1 Gate-level stuck-at fault model 28

2.2.2 Transistor-level stuck fault model 28

2.2.3 Bridging fault model 29

2.2.4 Delay fault model 29

2.2.5 Single Vs. Multiple fault models 30

2.3 DFT approaches for FPGAs 31

2.3.1 Scan-based DFT 31

2.3.2 Built-In Self-test (BIST) 32

2.4 BIST for FPGAs 35

2.4.1 BIST application procedure in FPGAs 36

2.4.2 BIST types for FPGA testing 38

2.4.3 Test time reduction 43

2.5 BIST design flow 45

2.5.1 Xilinx FPGA flow 45
2.5.2 JBits 47
2.5.3 Limitations of commercial FPGA tools 48
2.6 Verilog-to-Routing (VTR) Project 48
2.7 Conclusion 49
Chapter 3 Test and diagnosis schemes for novel FPGA 51
3.1 Overview of the targeted FPGA architecture 51
3.1.1 Cluster architecture 51
3.1.2 CLB architecture 53
3.1.3 Cluster size optimization 54
3.1.4 Switch box architecture 55
3.2 Test and Diagnosis Methodology 57
3.2.1 Test Pattern Generator (TPG) 58
3.2.2 Output Response Analyzer (ORA) 58
3.3 Test methodology for CLBs 59
3.4 Test methodology for crossbar 'Up' 61
3.5 Test methodology for crossbar 'Down' 64
3.6 Optimization of test configurations for crossbar 'Up' and CLB 67
3.7 Generalization of CLB and crossbar 'Up' joint testing 70
3.8 BIST structure for cluster 71
3.9 Switch box test methodology 73
3.9.1 Test configurations in phase 1 74
3.9.2 Fault detection and diagnosis in phase 1 77
3.9.3 Test configurations in phase 2 78

3.9.4 Fault detection and diagnosis in phase 2 80
3.9.5 Test configurations in phase 3 80
3.9.6 Fault detection and diagnosis in phase 3 81
3.10 Conclusion 82
Chapter 4 Test and diagnosis schemes for defect tolerant FPGAs 83
4.1 Defect tolerance in FPGAs 83
4.1.1 Software-based hardening 83
4.1.2 Hardware-based hardening 83
4.2 Redundancy in logic blocks 85
4.3 Test configurations for *Butterfly* LUT/CLB 88
4.3.1 Diagnosis in *Butterfly* LUT 88
4.4 Redundancy in interconnect 89
4.4.1 Fine Grain Redundancy (FGR) 90
4.4.2 Distributed Feedback (DF) 92
4.5 Conclusion 93
Chapter 5 94
Test automation and integration into FPGA CAD flow 94
5.1 BIST configuration flow 95
5.1.1 Placement constrain files 97
5.1.2 Routing constraints files 98
5.1.3 Integration into CAD flow 100
5.2 BIST Validation flow 102
5.3 Validation methodology 102
5.4 Validation flow 105

5.5 Fault diagnosis 107
5.6 Fault mapping 108
5.7 Conclusion 110
Chapter 6 111
Experimental evaluation and results 111
6.1 Impact of cluster-size on the FPGA testability 111
6.2 BIST simulation results 113
6.2.1 Results of BIST implementation for cluster 113
6.2.2 Results of BIST implementations for switch box 116
6.2.3 Test time optimization results using joint testing 117
6.2.4 Test time optimization results using partial reconfiguration 119
6.3 Impact of defect tolerant techniques on FPGA testability 121
6.3.1 Logical masking 122
6.3.2 Routability and defect avoidance 124
6.4 Scan-DFT results 125
6.5 Comparison between FPGA BIST and scan-DFT 129
6.6 Conclusion 130
Chapter 7 131
Conclusion and perspectives 131
7.1 Conclusion 131
7.2 Perspectives 135
Appendix A 137
Mesh of clusters FPGA architecture perspectives 137
The hierarchical FPGA 137

Efficiency of mesh of clusters FPGA: Area and routability 138

Appendix B 144

BIST integration tools 144

Constraint files (.desc) 144

Script files (.tcl) 145

Glossary 146

List of Publications 147

References 148

List of Figures

Figure 2.1: Classic mesh of cluster FPGA with connection box 21

Figure 2.2: Structure of a CLB and a LUT 22

Figure 2.3: Types of switches in FPGAs 23

Figure 2.4: Cluster with fully populated crossbar.....24

Figure 2.5: Mesh of cluster FPGA without connection box 25

Figure 2.6: Cluster with sparsely populated crossbar 25

Figure 2.7: Structure of a multilevel switch box.....27

Figure 2.8: Basic BIST structure 33

Figure 2.9: FPGA BIST structure 36

Figure 2.10: BIST structure and test sessions.....39

Figure 2.11: Circular BIST for FPGAs.....40

Figure 2.12: Configuration MUX 40

Figure 2.13: Interconnect BIST in FPGA 41

Figure 2.14: Online BIST in FPGA [Abramovici 1999] 43

Figure 2.15: Xilinx bitstream generation flow.....46

Figure 2.16: BIST for Xilinx FPGAs.....46

Figure 2.17: VTR Project flow for FPGAs.....49

Figure 3.1: Mesh of clusters FPGA 52

Figure 3.2: Cluster in a mesh FPGA with depopulated crossbars 52

Figure 3.3: Structure of a crossbar.....53

Figure 3.4: a) CLB and b) 4-input LUT in the cluster of a mesh FPGA 54

Figure 3.5: Structure of a Switch Box in a mesh FPGA 56

Figure 3.6: Example of a mini switch box with 4 inputs and 3 outputs 56

Figure 3.7: Logic block of comparison-based ORA cluster 59

Figure 3.8: Exemplary scenario of BIST 61

Figure 3.9: Sequence of test configurations for crossbar-up in cluster-size 4 63

Figure 3.10: Test configurations for crossbar 'Down' in a cluster of size 4 66

Figure 3.11: Test configurations sequence of Crossbar Up of cluster size 4 69

Figure 3.12: BIST structure in 4x4 FPGA mesh 72

Figure 3.13: Structure of a switch both in mesh FPGA 74

Figure 3.14: BIST structure in phase 1 showing paths under test (PUTs) for UMSBs testing 76

Figure 3.15 : BIST structures in phase 2 showing paths under test (in red) for testing Switch Box (SB) outputs 79

Figure 3.16: BIST structures in phase 3 showing paths under test (in red) for testing Switch Box (SB) inputs 81

Figure 4.1: a) Simple LUT-4 b) LUT-4 with TMR [Kyria 2009] 86

Figure 4.2: LUT-4 modified Butterfly design [Dhia 2013] 87

Figure 4.3: Fault tolerant voter [Ban 2010] 87

Figure 4.4: BIST structure for CLB/LUT 88

Figure 4.5: Fault masking example in *Butterfly* LUT 89

Figure 4.6: Crossbar 'Down' hardened with FGR technique [Amouri 2013] 90

Figure 4.7: Crossbar 'Down' hardened with FGR technique [Arwa 2013] 92

Figure 5.1: VTR Project CAD flow 96

Figure 5.2: Placement constraints file sample 98

Figure 5.3: Routing constraints file sample 99

Figure 5.4: BIST integrated into FPGA CAD flow 101

Figure 5.5: Cluster with SRAM [x:0] as primary inputs 103

Figure 5.6: 11:1 MUX as hierarchical structure made of 2:1 MUXes 104

Figure 5.7: BIST algorithm verification flow 107

Figure 5.8: Fault mapping between configuration bitstreams and fault lists 109

Figure 6.1: Fault coverage vs. Number of test configurations of a cluster 115

Figure 6.2: Comparison of joint and separate testing of CLB and crossbar 'Up' for various cluster sizes 118

Figure 6.3: Fault coverage vs. number of configurations for joint testing 119

Figure 6.4: Mesh of clusters FPGA with configuration port 120

Figure 6.5: Memory requirement for full and partial test reconfigurations of a cluster 121

Figure 6.6: Distribution of detected faults and the corresponding fault coverage for the basic cluster vs. number of test patterns 126

Figure 6.7: Distribution of detected faults and the corresponding fault coverage for the cluster with FGR vs. number of test patterns 126

Figure 6.8: Distribution of detected faults and the corresponding fault coverage for the cluster with DF vs. number of test patterns 127

Figure 6.9: Distribution of detected faults and the corresponding fault coverage for the cluster with *Butterfly* LUT vs. number of test patterns 128

Figure 6.10: Distribution of detected faults and the corresponding fault coverage for the cluster with *Butterfly* LUT, FGR and DF vs. number of test patterns 128

Figure 6.11: Comparison of fault coverage for the basic cluster with Butterfly and FGR. 129

Figure A.1: Hierarchical FPGA architecture 138

List of Tables

Table 3.1 Cluster I/Os for Different sizes.....55

Table 3.2 LUT configurations to propagate an input to output.....64

Table 3.3 Test Configurations of Logic Block and Crossbar Up for Cluster Size 4.....69

Table 3.4 Configurations and Test Patterns for a UMSB at 'Level 1'.....77

Table5.1: LUT configuration bits for propagating any of its inputs to the output.....105

Table 6.1 Cluster I/Os for Different sizes.....112

Table 6.2 UMSBs and DMSBs in a Switch Box for Different sizes.....113

Table 6.3 Results for Cluster BIST.....114

Table 6.4 Results for Switch Box BIST.....117

Table 6.5 Results for optimized Test Configurations.....118

Table 6.6 Results for 100% Fault Coverage of a Cluster.....121

Table 6.7 Results for 100% Fault Coverage of a Cluster.....122

Table 6.8 Logical maskign emulation results [Dhia 2013].....123

Table 6.9 Defect avoidance and routability results [Amouri 2013].....124

Table 6.10 Results for Scan-design DFT.....129

Chapter 1

Introduction

1.1 Motivation

The tremendous development of CMOS technology has enabled an increasing integration density according to Moore's Law [Moore 1998]. Shrinking feature size and increasing complexity in modern electronics enables to follow the trend of compact and high performance devices. However, this evolution trend is being slowed down due to increasing overall design, verification and manufacturing costs while approaching the physical limits. This very high density integration close to the physical limits generates numerous imperfections leading to a new mandatory paradigm: produce integrated circuits able to tolerate all possible physical defects and variations or transient and intermittent faults that may occur during the fabrication steps or projected for the life time duration of the chip. To maintain higher yield figures, one of the challenges is to find a way to design and use fabricated circuits while tolerating physical defects and variations spread all over the chip.

Test for manufacturing defects and variations have always been considered as an integral part of the IC development process. Therefore, test procedures were constantly improved and adapted to keep the pace with the design and validation IC advancements and vice versa. Efficient test equipments and methodologies are not only required to improve the device reliability and yield but also to reduce the test cost associated with it. Test cost is different for different devices. For some it may account for 70% of the total manufacturing cost while for some it may exceed the manufacturing cost [S. May 2006]. Test cost management involves many variables such as manufacturing equipment efficiency, test time, device application domain etc. Moreover, test solutions do not scale linearly with the process technology, device size and functionality [ITRS 2013].

FPGA components have proven at the best their capabilities in prototyping and emulation phases of the complex digital systems development process. Many multi-core SoCs are currently being emulated by using some of the latest FPGAs platforms,

exploiting very well FPGAs' high performance and low power features. In addition to this, due to their low development cost and reduced time-to-market, SRAM-based FPGAs are widely used in many applications from networking to telecommunications and even in space applications for which specific FPGAs such as antifuse-based and radiation-tolerant are developed [AFSoC; RTAX; Seifert 2006]. FPGA's main feature, the reconfigurability, is not only exploited for the flexible design of multi-core chips but also during the test for manufacturing defects. In FPGAs, defect locations can be avoided by reconfiguring the application on fault-free resources provided the fault location is known. Diagnosis of a given circuit comprises both the detection and the location of faults. Therefore, diagnosis is critical for the components and devices which rely on the defect bypass mechanism during their reconfiguration phases.

Testability is often discussed in terms of test cost and fault coverage. In case of FPGAs, a major issue in test cost is the reconfiguration time which makes FPGA testing expensive compared to ASIC. To program an FPGA for a given application (that can also be a testing application), dedicated configuration bits are loaded into the FPGA memory array (SRAM cells). These configuration bits are usually stored on a memory located outside the chip and grouped into frames of a certain bit-width. For loading an application, these bits are accessed one frame at a time and written into the configuration SRAM cells available inside the FPGA. Usually test cost is evaluated in term of test time (i.e. (re)configuration time) and the memory size required to store the configuration bitstream. Therefore, targeting the highest fault coverage through exhaustive testing may become very costly for FPGAs, sometimes prohibitive.

Similar to digital ASICs, FPGAs are tested for structural faults (i.e. stuck-at, transient, bridging and delay faults) using DFT techniques such as scan-based design and BIST. These DFT approaches have become quite mature over the last ten years. However, optimization of these approaches for new and emergent device architectures as well as the test time reduction is still considered as the main challenge. The main purpose of the DFT approach is to improve the testability of a given circuit which ultimately improves the system's reliability.

- In traditional scan-design based DFT, flip-flops from the sequential logic blocks of the FPGA are chained into shift registers by configuring and adding one multiplexer to each data input of every flip-flop. Test data input is 'scanned-in' through 'scan-input' port and connected to the first multiplexed flip-flop, and the normal logic output is connected to the next input of multiplexed flip-flop, etc. Multiplexer select input is used to select the 'test' or 'normal' input in 'test' or 'normal' mode respectively and is connected to the 'scan-enable' at primary input.

The output of the last multiplexed flip-flop is 'scanned-out' to one of the primary output, or to a dedicated 'scan-out' pin. As a result, a 'scan-chain' is formed in which all flip-flops are connected in series, making the design easily controllable and observable from primary inputs/outputs. Using scan-design based approach; up to 100% stuck-at fault coverage can be achieved [Bushnell 2000]. Implementation of this approach can be fully automated including scan-chain insertion, test pattern generation, fault simulation and fault coverage analysis. However, the requirement of external resources limits to perform system-level test. Moreover, scan-design DFT utilizes deterministic algorithm for vector generation targeting a 1-detect fault set, thus requiring considerable amount of test time to cover for all faults.

- In Built-In Self-Test (BIST) approaches, a part of the FPGA can be configured to test some other parts. A single BIST operation is usually performed in two test sessions. In the first session, some FPGA logic blocks are configured to generate test patterns and others to analyze the output response while some other parts are configured as circuit under test. The resources under test go through a number of successive configurations so that they can be tested in all possible modes of operation. For each configuration, specific test patterns generated by pattern generator are applied to the parts under test and the results are analyzed at the end of each test sequence. At the completion of all test configurations, the second test session starts in which FPGA logic blocks swap their roles and complete the test of the whole FPGA.

BIST has some major advantages over scan-design based DFT for FPGA testing. BIST utilizes device reconfigurability for fault detection. BIST configurations are removed at the end of the test mode and the FPGA gets into a normal application mode on the same resources. Hence, BIST offers an indigenous property of avoiding any area or performance penalty for testing FPGAs. Moreover, parallel testing can be performed to achieve further test reduction time. Similarly, BIST allows performing all level testing from wafer to system-level [Stroud 2002].

Although BIST is a generic technique, test configurations are architecture specific. For that reason, different BIST approaches are developed for different FPGA device architectures. Nowadays, modern FPGA architectures are mainly organized in clusters of configurable logic resources connected together by depopulated interconnect. However, cluster architecture organization and size versus inter and intra-cluster interconnect architectures is an ongoing optimization process, as it severely impacts the routability, area saving, testability and the overall robustness of a given FPGA [Kuon 2007].

Therefore, any improvement in FPGA architecture to deal with complexity, performance and space, must also be evaluated in terms of testability. For that purpose, BIST schemes have to be developed to target optimized test cost and maximum achievable fault coverage.

As mentioned earlier, defect avoidance in FPGAs can be achieved utilizing its reconfigurability in addition to testability. Most of the defect tolerant schemes resort to redundancy and can be classified into software-based and hardware-based techniques [Gusmão 2004; Kastensmidt 2006].

Software-based techniques avoid defective resources by means of place-and-route tools which map a given design around previously detected defects. Hence, the efficiency of the software approaches relies on the performance of such tools as well as the availability of the free resources.

Hardware-based techniques employ modifications in the basic architecture. In some cases, extra hardware resources are added, providing redundancy at different granularity levels. In some other cases, architecture optimization is performed to automate the configuration bits shift mechanism. Such hardening techniques ultimately modify the FPGA architecture. If the test scheme for this FPGA is architecture dependent, modification in the test scheme will be needed anytime architecture is modified.

Most of the commercial FPGAs have dedicated CAD tools to perform emulation/simulation. For example, Xilinx has *ISE Design Suite* (Integrated Synthesis Environment) which provides a complete flow from RTL to configuration bitstream for its wide range of FPGAs [Xilinx]. Similarly, *Quartus* is dedicated for Altera FPGAs [Altera]. Atmel FPGAs can be used with conventional EDA tools available in the industry such as *Mentor's Precision Synthesis* for RTL to synthesis process. Then *IDS* (Integrated Development System – Atmel) is used as EDA tool to place and route synthesized designs [Atmel]. For academic and research purposes, *Verilog-to-Routing Project* (VTR) has developed an independent flow to configure a design for customized FPGA architectures [Rose 2012]. It includes multiple open source software such as *ODIN-II* for synthesis, *T-VPack* for cluster packing and *VPR* for place and route. VTR Project has gained a lot of trust in academia and industry due to its versatile platform and ability to integrate various classic and in-house developed tools in the flow. For this reason, VTR Project flow is often used in development of new FPGA architectures, specifically for the verification of their routability and testability.

The motivation of this PhD work is to develop test and diagnosis techniques for a new cluster-based mesh FPGA architecture composed of a novel hierarchical multilevel

interconnect topology. This new interconnect architecture promises improved routability and significant area and power reduction. Moreover, exploitation of FPGA architecture for test time reduction and extension of the proposed test and diagnosis techniques for the various defect tolerant FPGAs are the key features of this work.

The work done in this thesis is a part of a project named *Robust FPGA* [RobustFPGA]. This project is a collaboration of [LIP6], [ParisTech], [TIMA] and [FlexRAS] which deals with the development of defect tolerant FPGA architecture specifically based on multilevel interconnect topology. It involves the development of FPGA architecture generator, defect tolerance schemes in FPGA, testability and routability.

1.2 Key issues and contributions of the PhD thesis

The key issues and the main contributions addressed in this dissertation are the followings:

1. Development of test and diagnosis schemes for stuck-at and bridging faults in logic and interconnect resources of the FPGA containing hierarchical interconnect topology.
2. Optimization of the proposed test and diagnosis schemes for test time reduction avoiding area overhead, hardware modification or loss in diagnostic resolution.
3. Analysis of the impact of various architectural parameters on the FPGA testability and the respective test cost.
4. Extension of the proposed test and diagnosis techniques to the FPGAs architectures incorporating defect tolerant techniques at logic and interconnect level.
5. Development of tools for automated test and diagnosis schemes implementation and their integration into the standard design flow for bitstream generation.

1.3 Thesis organization

The thesis manuscript is organized as follows:

Chapter 2 gives a brief overview of the techniques used for FPGA testing. It also covers the challenges and limitations in development of FPGA test methodology and generalization. The proposed test and diagnosis techniques targeting the novel FPGA architecture are detailed in chapter 3 where the FPGA architecture is also described. The techniques developed for test time optimization are also explained in this chapter. Chapter 4 deals with the testability of the FPGA containing several defect tolerant structures. The impact of defect tolerance on the FPGA testability is evaluated. The automation and integration of the proposed test schemes into the standard CAD flow is given in chapter 5

which also discusses the fault mapping, scalability and diagnostic resolution of the schemes. Chapter 6 manifests the results of a set of experiments carried out to validate the proposed techniques. Chapter 7 summarizes the key contributions of this thesis and concludes the work with possible future directions and ideas.

Chapter 2

Test and Diagnosis of FPGA: State of the art

Deep submicron technology has enabled the development of compact and high performance devices at a relatively lower manufacturing cost. However, the devices have become more vulnerable to manufacturing defects which affects the reliability of the device. Consequently, testing a device for manufacturing defects has become a mandatory step in the development flow of an IC. The test cost of a device mainly depends on its testability which is defined as the degree of access to all resources of a given architecture, to detect any modeled defect. For highly integrated complex designs e.g. FPGAs, testing is not a straightforward task. Exhaustive testing of an FPGA to achieve 100% fault coverage is very time consuming. On the other hand, reprogramability of an FPGA, considered as a key feature that makes it prominent in nowadays applications, adds more complexity to its testing phases. As FPGA can be configured in a number of ways, it is very important to also test it in all possible configurations. Therefore testing an FPGA becomes a challenge.

FPGA test cost is related to the capability of the test mechanism to detect embedded faults. Most of the test mechanisms developed for FPGAs are architecture dependent. The main issues in FPGA testing are to achieve 1) generalized and scalable test mechanism for any FPGA array size, 2) test time reduction 3) integration of test mechanism into FPGA CAD flow so that the standard tools can be used to implement and verify the test coverage. In this chapter, we will discuss the overview of the FPGA architecture, fault models and state-of-the-art test techniques for FPGAs and the methods to reduce the test time presented in the past.

2.1 Overview of the FPGA architecture

Modern FPGAs vary very much in their architecture depending on their application domain. Mostly, it is the interconnect topology between logic blocks which makes them

different. Since the interconnect resources make up to 80% of the total FPGA area [Kuon 2007], trade-off between interconnect area and routability has been a very important issue in FPGA research. In the following, we describe the development of the basic FPGA architecture and the key structures of the FPGA building blocks.

2.1.1 Classic mesh FPGA architecture

In a classic mesh-based FPGA, a number of logic elements are grouped together forming a cluster and the clusters are arranged in a grid surrounded by vertical and horizontal routing channels. The connections between clusters are made through switch and connection boxes. The switch and connection boxes make the overall interconnect structure in the FPGA as shown in Figure 2.1. A connection box is used to connect a given cluster input and output pins to the adjacent routing channels while a switch box provides connection between horizontal and vertical routing channels. This is also called island style architecture as the logic blocks are surrounded by fixed interconnect wires. Most of the Xilinx and Altera FPGAs use island style architectures [Betz 1999].

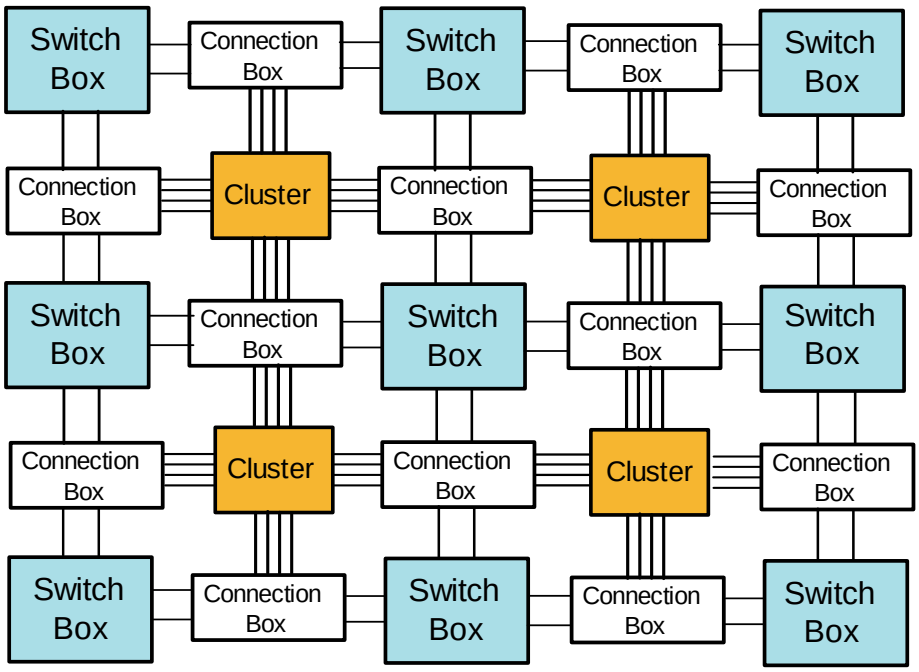


Figure 2.1: Classic mesh of cluster FPGA with connection box

The reconfigurability of FPGAs is a result of its re-programmable architecture. There are two main configurable components in FPGA architecture. 1) Logic blocks and 2) interconnects.

Configurable Logic block (CLB): A cluster may contain several CLBs implementing logic functions. Figure 2.2 represents a CLB with a 2-input LUT (LUT-2), a Flip-Flop

and a multiplexer 2:1 to select either the combinational or the sequential path. An LUT is based on SRAM cells followed by a series of MUX2s. The configuration bitstream (cf. Data input in Figure 2.2) is loaded in to the SRAM, while the LUT multiplexers (MUXes) implement the logic function, according to the values of inputs I_1 and I_2.

Usually Data (configuration bits) are loaded in the FPGA using Strobe signals which come from bitstream 'Configurator' or 'Loader'. The SRAM cells in logic blocks and interconnect are arranged in rows and columns. Each SRAM bit in a row receives a vertical Data signal and horizontal Strobe signal. The Data bits are loaded in to the SRAM cells of a row only when the Strobe signal for that row is high. In this way, Strobe signals are used to validate the correct SRAM cell to be loaded for the given configuration [Pervez 2011].

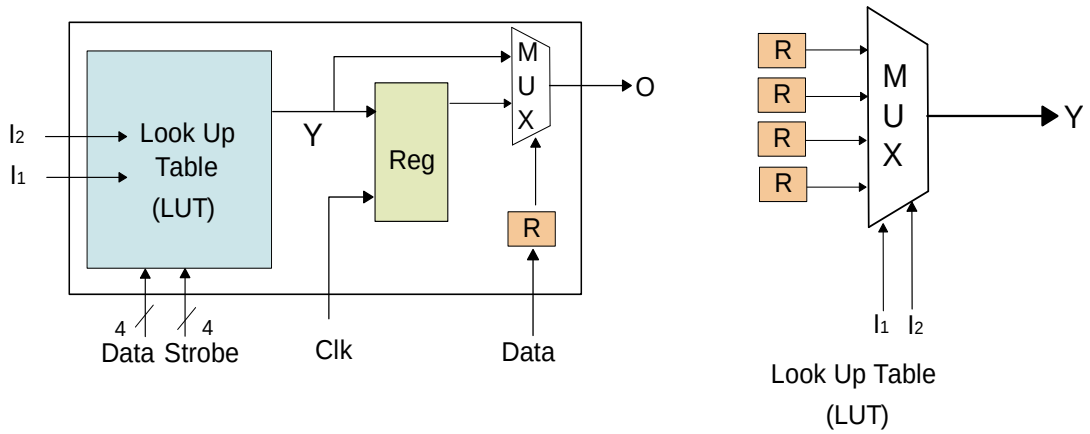


Figure 2.2: Structure of a CLB and a LUT

Interconnect: Connection boxes and switch boxes in the FPGA contain programmable switches which connect CLBs to the routing channels. The programmable connections inside the switch elements are called Programmable Interconnect points (PIPs). PIPs are implemented using combinations of programmable switches. Usually there are three kinds of switches found in the FPGAs as shown in Figure 2.3. In modern FPGAs, interconnect structures are made of multiplexers where 'select' inputs of the MUXes come from configurable SRAM cells. Configuration bits are loaded into the SRAM which selects the MUX input signals thus establishing the path through the interconnect structure. Pass transistors are bidirectional and used to connect wires whereas a tri-state buffer is used as unidirectional switch and is made of five transistors and a single SRAM cell.

2.1.2 Variations in FPGA Interconnect

Interconnect topology has been a critical factor in the development of new FPGA architectures. Optimization techniques for a lesser area with better routability has been presented in [Kuon 2008; Marrakchi 2009; Lin 2010]. In clustered FPGAs, the area

occupied by the interconnect resources depends on the way the routing signals pass through switch boxes and to the LUT inputs in the cluster. For research and academic purposes, Verilog to Routing Project (VTR) [Betz 1999; Rose 2012] has been developed for FPGAs where fully populated crossbars are used as interconnect in switch boxes and also inside the clusters. Figure 2.4 shows a structure of an intra-cluster interconnects with fully populated crossbar. It connects any input of the cluster to any input of the Configurable Logic Block (CLB). In addition to cluster inputs, CLB feedbacks are also fed to the inputs of each of the crossbars since it is fully populated. A Large number of switches/multiplexers are required to implement a fully populated crossbar. This architecture is simple and provides high degree of internal routability but at the cost of large area overhead and longer propagation delay. Also, it does not take advantage of the logic equivalence of the logic blocks which means that every logic block contains identical number of LUTs and has the capacity to perform equivalent logic functions [Marrakchi 2009]. Therefore, this architecture induces a significant area overhead in the case of a large number of logic blocks within the cluster. Some commercial FPGAs such as Stratix™ family from Altera are also made of fully populated inter and intra-cluster crossbars.

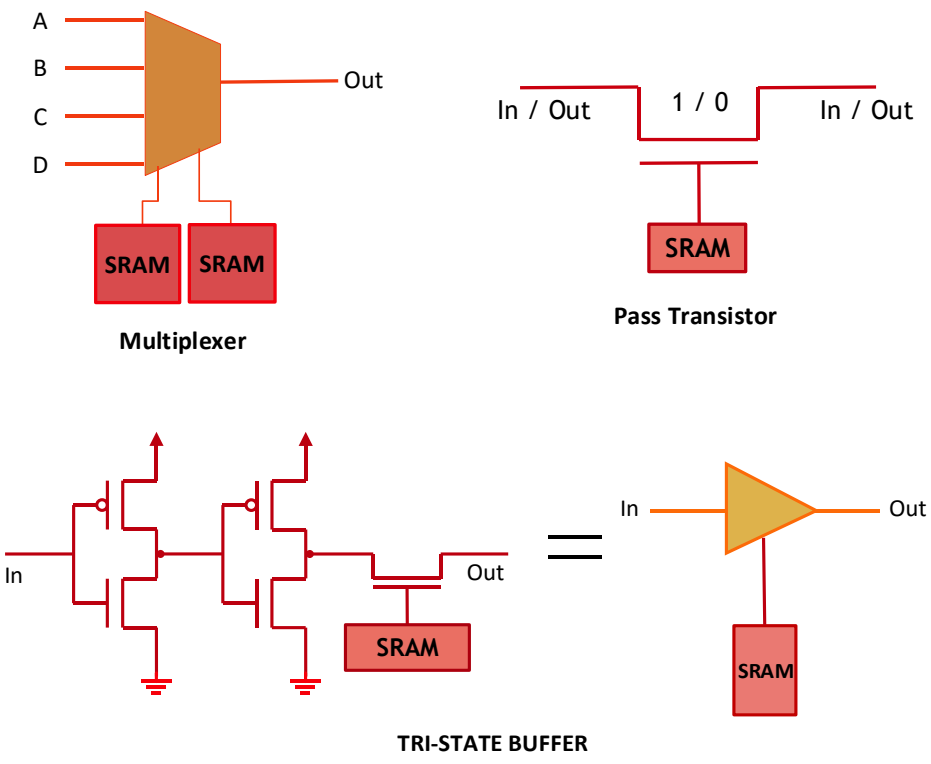


Figure 2.3: Types of switches in FPGAs

An improved *VPR style* interconnect was proposed in [Lemieux 2004] in which sparse/depopulated intra-cluster crossbars were used. In this architecture, an area saving of 10-18% was achieved by optimizing connection boxes and intra-cluster crossbars separately.

In [Feng 2007], crossbars in inter and intra-cluster interconnect are jointly optimized saving up to 28% of area. In this case, a full crossbar is used to connect local CLB feedback to the LUT inputs thus limiting the use of a large number of logic blocks in a single cluster.

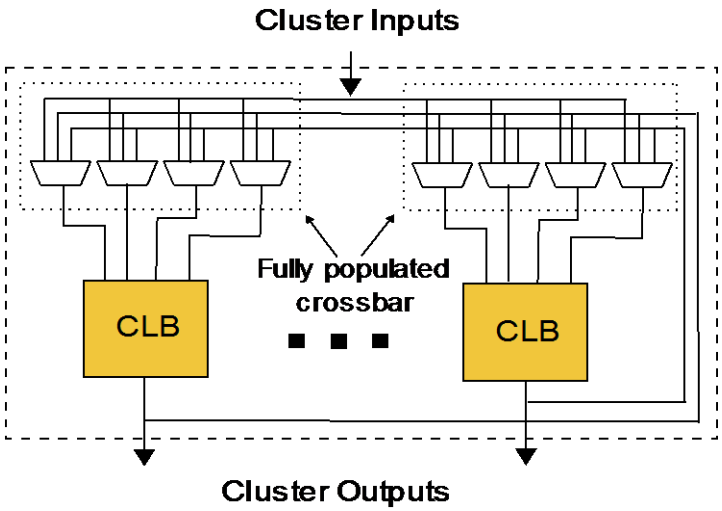


Figure 2.4: Cluster with fully populated crossbar

In modern FPGAs, the trend of having a large number of logic blocks in the cluster is increasing [Ahmed 2004]. The motivation behind including large number of logic blocks in a cluster is to reduce the interconnect area and to maximize the utilization of logic resources with an increased intra-cluster routability.

In Xilinx Virtex™ family FPGAs, routing channels are directly connected to the input multiplexers of the logic elements avoiding connection boxes aiming at reducing the interconnect area. Figure 2.5 shows the structure of mesh of clusters FPGA where cluster are directly connected to switch box. However, in most of Virtex™ FPGAs, fully populated crossbars are used as interconnect structures.

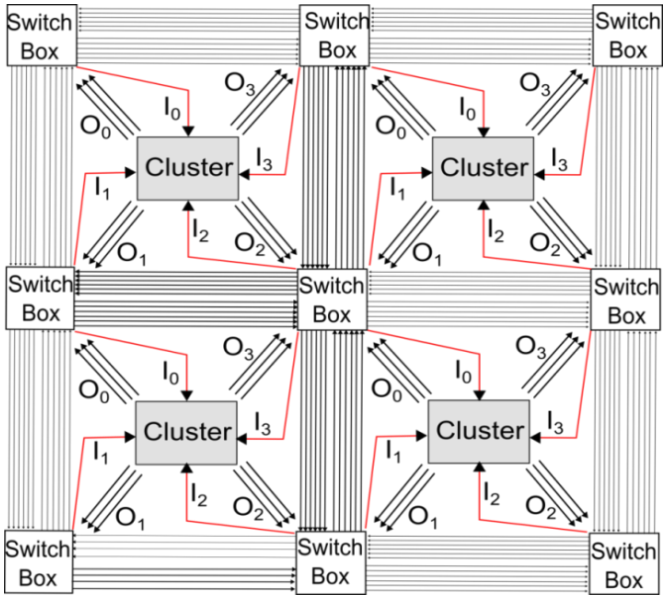


Figure 2.5: Mesh of cluster FPGA without connection box

2.1.3 Multilevel mesh FPGA architecture

In [Marrakchi 2009], an efficient mesh of clusters architecture is proposed in which a depopulated (sparsely) crossbar is used to connect external inputs and the local feedbacks to the LUT inputs. An example of such a cluster is shown in Figure 2.6. When using sparsely populated crossbar, the inputs of a cluster are uniformly distributed among all inner CLBs such that each CLB input connects to some specific inputs of the cluster/crossbar.

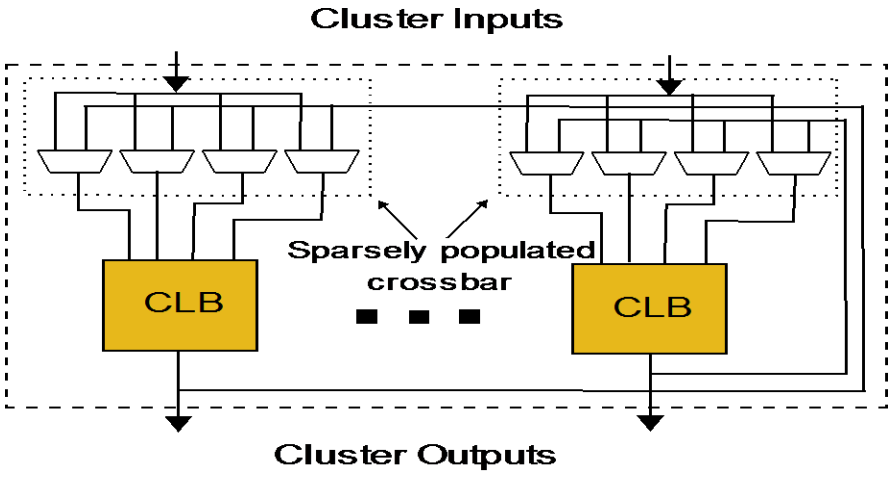


Figure 2.6: Cluster with sparsely populated crossbar

Using sparsely populated crossbars may affect the routability of the FPGA for a given application as it provides lesser number of possible connections among cluster inputs and CLBs. For that reason, application may require more FPGA clusters for its

implementation. However, using sparsely populated interconnect topology requires smaller multiplexers as compared to fully populated crossbars which can save considerable amount of chip area. Usually, a trade-off between routability and the area consumption is carried out by implementing several benchmark circuits for a given FPGA.

Extending the same concept of sparsely populated crossbars to the switch box, two unidirectional networks are used to connect routing channels together and to the CLBs, forming a novel hierarchical interconnect topology. This novel FPGA interconnect was developed by [Marrakchi 2010]. This topology comprises two levels of interconnect. Level 1 contains depopulated crossbars providing connection among clusters whereas connection between switch boxes (routing channels) is established through depopulated crossbars forming level 2 of the hierarchical interconnect. Figure 2.7 shows the structure of such a multilevel switch box.

Inputs of the switch box are uniformly distributed among crossbars at 'Level 2'. The number of crossbars at 'Level 2' is determined by the value of FPGA Channel Width (CW) which is defined as the number of unidirectional wires connecting two switch boxes together. Therefore, the number of crossbars at 'Level 2' becomes half of the CW as each crossbar connects only one of its outputs to the adjacent switch box. Similarly, output of the adjacent clusters are distributed among crossbars at 'Level 1' which provide signals s1, s2, s3 to be connected to the crossbars at 'Level 2'. These crossbars also provide feedback paths to the cluster by connection its signals s1, s2, s3 back to the cluster inputs via another set of crossbars at 'Level 1'. This set of crossbars then uniformly distributes the connections from a switch box to its adjacent clusters.

Experimental results using this novel interconnect topology are presented in [Amouri 2013] which show that the area of multilevel mesh FPGA is decreased by 42% compared to the cluster-based VPR-style architecture without losing much routability. Moreover, developers of this architecture also claim that the proposed multilevel interconnect topology performs better as compared to the FPGA having tree-based interconnect. A tree-based architecture [Farooq 2008] contains a depopulated intra-cluster crossbar and unifies two unidirectional networks; a downward network based on *Butterfly-Fat-Tree* topology and an upward network using hierarchy. Such a tree-based architecture gives 56% of area saving compared to cluster-based VPR-style architecture. This better area saving of tree-based (56%) compared to the novel multilevel mesh-based architecture (42%) is compensated by the simplicity of layout generation in case of mesh-based [Amouri 2013].

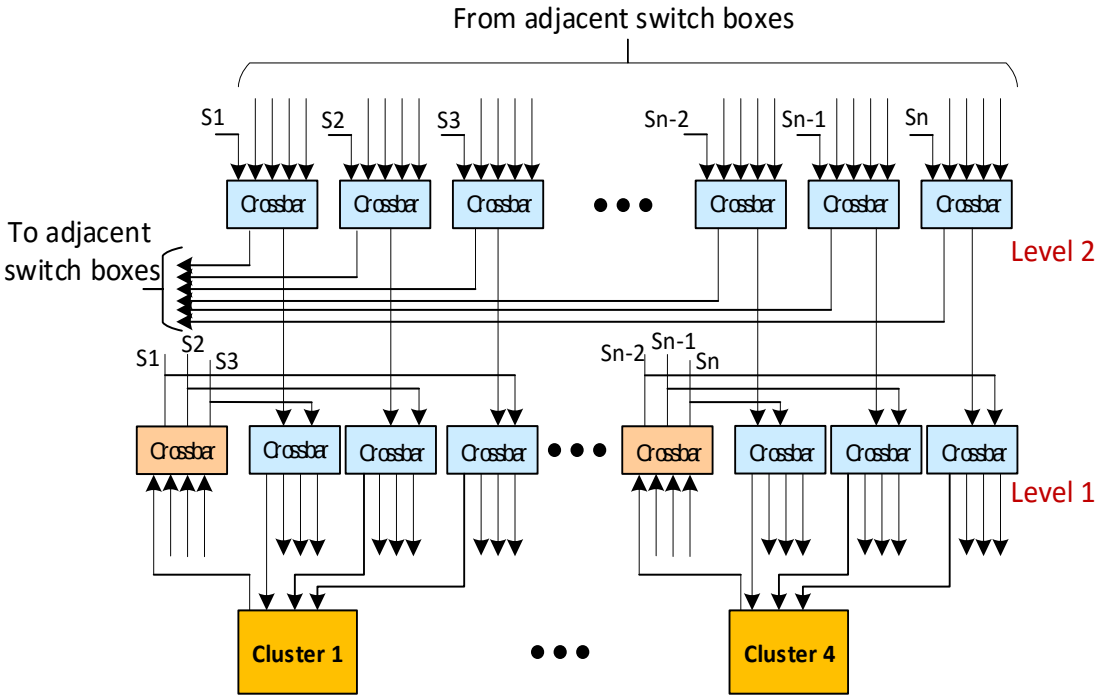


Figure 2.7: Structure of a multilevel switch box

This multilevel FPGA architecture has been chosen for this work and in the next chapter we will discuss in details the internal structure of the cluster and switch box. These details are necessary when describing the test mechanism developed for this FPGA. Thus, we have made the choice to talk about that later.

2.2 Fault modeling

Fault is a logical effect of a defect - a physical imperfection that may occur during the fabrication/manufacturing process or develop during the lifetime of a device. Depending on the cause of defects, faults can be broadly classified as temporary or permanent.

There are two main types of temporary faults: 1) transient and 2) intermittent.

Transient faults may be caused by radiation or power supply fluctuation or electromagnetic interferences, and so on and they last for a limited duration of time. Usually transient faults do not cause damage to the physical hardware thus we cannot talk about repairing a unit affected by transient faults.

Intermittent faults are recurring faults that reappear periodically at regular or irregular intervals; when environmental conditions meet again. Such faults can be caused by some defective component in the circuit, loose connections or poor design. Some

intermittent faults may eventually become permanent due to deteriorating component or device aging.

Permanent faults are caused by irreversible physical defects in the circuit. In most cases, these types of defects occur during the manufacturing process. Permanent defects may also occur due to aging or wear out during normal usage of the device.

Test schemes and analysis tools are developed to effectively detect faults which require fault models for the emulation and analysis of the physical defects. It is important for a fault model not only to reflect the actual behavior of the defect but also to ease the computational effort during the fault simulation process. Therefore, they have to be simple and portable and as much as possible independent of technology. In this work, we will focus on the detection and diagnosis of the permanent faults in the FPGA. Classic FPGA testing utilizes conventional models for permanent faults which include gate-level faults, transistor level faults, bridging faults model and delay faults model.

2.2.1 Gate-level stuck-at fault model

This model reflects the effect of a permanent defect at the gate input and output as a fixed logic value irrespective of the value actually driven at that node. A signal shorted to VDD (power) appears as permanently high i.e. stuck-at 1 (SA1) and a signal shorted to VSS (ground) appears as permanently low i.e. stuck-at 0 (SA0). To be able to detect stuck-at faults, all considered faulty nodes must be controllable and observable. Testing of gate level stuck-at faults involves the application of a set of test vectors at primary inputs followed by the propagation of the fault response to the primary outputs. The output response is compared with the expected fault free output.

2.2.2 Transistor-level stuck fault model

This model features the transistors to be either stuck-short (stuck-on) or stuck-open (stuck-off). Stuck-short can be emulated by a permanent short between source and drain of a transistor or by connecting transistor gate to a logic '1' (VDD) for NMOS transistor (to logic '0' (VSS) for PMOS). Similarly, stuck-open can be emulated by disconnecting transistor from the circuit or by connecting transistor gate to a logic '0' for NMOS (to logic '1' for PMOS). In case of the stuck-short fault in the transistor, there is a steady flow of current from VDD to VSS whereas in case of fault free transistor, there is only normal leakage current. For this reason, analysis of the steady-state power supply current (IDDQ) is used to detect stuck-short faults in the circuit.

In case of stuck-open fault in a transistor, there is no path for the current from VDD or VSS to the output node. Consequently, the output node retains its previous logic value. To detect stuck-open faults, a sequence of test vectors is applied to store an opposite logic value from that of the fault free circuit at the faulty gate output.

2.2.3 Bridging fault model

During fabrication processes, over and under-etching of metal layers can cause defects which include open and short between wire segments. In case of open wire segments, signals cannot propagate to the instances causing similar behavior as in the case of gate/transistor-level stuck-at faults. For this reason, open wire faults can be detected using test vectors obtained by gate-level and transistor-level fault models. Short wires faults are usually termed as bridging faults. To detect bridging faults, two fault models are usually used.1) wired-AND/wired-OR bridging fault model and 2) the dominant bridging fault model.

In the wired-AND model, the logic value provoked by the bridging fault on both wires is a '0'. This works like an AND gate in which the gate output value is determined by a logic '0' at any input, thus called as 0-dominant bridging fault. Likewise, in the wired-OR the logic value on both wired is determined by a '1' on either wire. It is known as 1- dominant bridging fault. In the dominant bridging fault model, the strongest drive wire determines the logic value at the destination end of the shorted wires.

For a given set of test vectors, faults associated with the particular bridging fault model are detected by emulating two faults during fault simulation. This is analogous to emulating both stuck-at-0 and stuck-at-1 for a gate-level fault or stuck-on and stuck-off for a transistor-level fault.

In order to detect bridging faults, opposite logic values should be applied on the two wires. Therefore to detect the wired-AND/OR bridging faults, there are two solutions: 1) monitoring both wires at primary outputs by applying any of the test vectors (01 or 10) and 2) monitoring only one of the wires at a primary output by applying both vectors (01 and 10). However, only the first solution can detect the dominant bridging faults [Stroud 2002].

2.2.4 Delay fault model

Delay faults exist in the circuit which performs its function correctly but operates at a lower frequency than required. This is usually caused by the under or over-etching during IC fabrication processes producing much narrower channel width or much longer channel

length in the transistors. In order to determine if any path in the circuit fails to operate at the required frequency, delay fault testing is performed on the paths between flip-flops outputs and primary inputs to the next level of flip-flops, and flip-flops outputs to primary outputs through the combinational logic. In delay fault testing, a sequence of two test vectors is applied. A specific output value on the path through a combinational logic is set-up by the first vector and then a transition is produced at the same output through the path by the second vector. A delay in transition means there is a delay fault. Delay faults are not considered in this work but are planned as a future work.

2.2.5 Single Vs. Multiple fault models

In practice, multiple defects can occur in a device. However, most of the modern fault analysis and simulation tools still use single fault model. In a single fault model, it is assumed that only one fault exists at a time in a given circuit or a part of a circuit. For a circuit with *N* gate inputs and outputs, *2N* faulty circuits can be emulated using single fault model. Multiple fault model is an extension of the single fault model where more than one fault simultaneously exist in the circuit. Therefore, in a circuit with *N* gate inputs and outputs, there will be 3^N-1 different faulty circuits under the multiple stuck-at gate level fault model.

In the past, several researchers have worked for modeling multiple faults [Kim 2002; Zhao 2010]. Most of these works proposed algorithms to model any given multiple faults as a single fault. Such as [Kim 2002] that proposed the insertion of n+3 modeling gates in the targeted paths of the circuit, where n is the multiplicity of the targeted faults. The purpose is to model the multiple stuck-at faults as single stuck-at faults keeping the modeled circuit functional equivalent to the original circuit. A two input gate (AND/OR) is inserted for each faulty line considered in a circuit. The controlling input for this gate is the same as the faulty value stuck-at the line. AND gate is inserted for the line having stuck-at 0 and OR gate is inserted for the line having stuck-at 1. To feed the second input of these inserted gates, an n-input AND gate is inserted. To keep the functional equivalence of the circuit, NOT gate is also added in line with the second input for the lines having AND gate inserted. Thus, a large number of gates is required to model multiple faults as single faults which is clearly not feasible for all or many multiple faults in the circuit. Therefore, multiple fault model is not only complicated but it is extremely expensive in terms of time required for fault analysis especially for larger circuits. Moreover, studies have shown that a large percentage of multiple faults can be covered by single fault tests [Hughes1986; Bushnell 2000; Stroud 2002].

Gate-level stuck-at faults can be analyzed more efficiently as compared to other fault models. But it is usually perceived that bridging and transistor level faults should also be taken into consideration as they are critical too. Recent studies show that a certain set of test vectors obtained by using gate-level stuck-at fault model are as effective in detecting transistor level stuck-open and bridging faults as test vectors developed specially for these faults models. These set of test vectors are referred to as *N-detect test set* in which a single fault is detected multiple times, each time by a different test vector of the set. These *N-detect set* of test vectors are usually developed using elementary logic gates (e.g. AND, OR, NOT, NAND, NOR) representation of the device. The *N-detect set* of test vectors developed using pin-faults of functional model such as XOR gate, multiplexer or flip-flop are considered as less effective in detecting faults.

Using *N-detect test* set is very efficient in test methods such as BIST where test pattern generator can easily produce a long test sequence. As a result, it increases the possibility of detecting each fault multiple times using different test vector. Although it will increase the fault evaluation time, the simulation time in BIST is negligibly small as compared to the configuration time. Therefore, in BIST, better results can be obtained using single stuck-at fault model with N-detect test set as compared to multiple fault for different fault models [Stroud 2002].

In this dissertation, single gate level stuck-at and bridging fault models are used for the development and verification of the proposed test methodology. Another advantage of gate-level stuck-at fault model is that it offers better fault collapsing as compared to transistor-level stuck faults which reduces the fault simulation time.

2.3 DFT approaches for FPGAs

The goal of DFT is to improve the testability of a circuit. Usually it is achieved by inserting extra hardware into the circuit to improve the controllability and observability of the internal nodes in the circuit under test, thus shortening testing time. There are two main types of DFT techniques that are explored for FPGAs; 1) Scan-based DFT and 2) BIST.

Both of these techniques are mature and well developed. Each has its own advantages and drawbacks which are explained in the following paragraphs.

2.3.1 Scan-based DFT

In scan-design based DFT, flip-flops in a sequential logic blocks of the FPGA are modified into shift registers by adding a multiplexer to the input of each flip-flop. Test

input is 'scanned-in' through scan input port and connected to one of the input of the first multiplexed flip-flop, and the normal logic input is connected to the other input of multiplexed flip-flop. Multiplexer select input is used to select the 'test' or 'normal' input in 'test' or 'normal' mode respectively and is connected to the 'scan-enable' at primary input. The output of the first (and next) multiplexed flip-flop is then connected to the test input of the following flip-flop or 'scanned-out' to the primary output in case of the last flip-flop. As a result, a 'scan-chain' is formed in which flip-flops are connected in series, making the design easily controllable and observable from primary inputs/outputs. Several scan designs exist: Full scan style design where all flip-flops of the circuits are chained up in the scan chain or partial scan where only those flip-flops that contribute to an increase of the fault coverage are considered.

Using scan-design based approach for FPGAs allows obtaining up to 100% stuck-at fault coverage [Bushnell 2000]. Implementation of this approach can be fully automated including scan-chain insertion, test pattern generation, fault simulation and fault coverage analysis. However, in FPGA testing, scan-based DFT does not fully exploit the regular structure of an FPGA for test time reduction. Scan-based DFT requires long application time due to the serial application of the test vectors and retrieval of test results. Several methods have been proposed to reduce the number of test configurations required to load onto the FPGA in Scan-based [Doumar 2000; McCracken 2002]. Most of the proposed methods require added hardware in the FPGA architecture. As a result, it not only costs in terms of area but also requires the modification in basic FPGA architecture. In [McCracken 2002], fast testing is made possible for FPGAs in which SRAM is implemented by means of shift registers, not as classical RAM. In [McCracken 2002] another technique is presented to speed up testing of switch matrix where a series of feedback shift registers is added to each switch matrix of the FPGA.

Although scan-based DFT is fully automated and give approximately 100% fault coverage, it is not considered suitable for testing FPGAs especially at system level, as it is difficult to apply test vectors at the system frequency. Moreover, the requirement of an available ATE and the area and performance penalty for test at-speed are also considered as the limiting factors in this regard. Despite these drawbacks, we will implement the scan-based DFT approach for testing logic blocks of the FPGA. The main reason is to have a reference for comparison with the proposed BIST schemes.

2.3.2 Built-In Self-test (BIST)

BIST is a general technique in which a circuit is designed in such way that it can test and identify itself as faulty or fault-free. In most of the cases, BIST is not only used for

fault detection but also to locate faults in the circuit (diagnosis). As discussed in previous section, a set of test vectors is required to detect faults and ensure high fault coverage. In BIST, these test vectors are produced inside the chip whose part/s is designed as Test Pattern Generator (TPG). Similarly, Output Response Analyzers (ORAs) are designed inside the chip and its main purpose is to analyze the responses for fail/pass indication. In addition to this, a specific circuit known as test controller is required to activate/deactivate the BIST. Among the roles of a BIST controller we can identify: initialization of TPG/ORA, indication of start/end of the test sequences, retrieval of ORA results etc. A basic BIST architecture is shown in Figure 2.8.

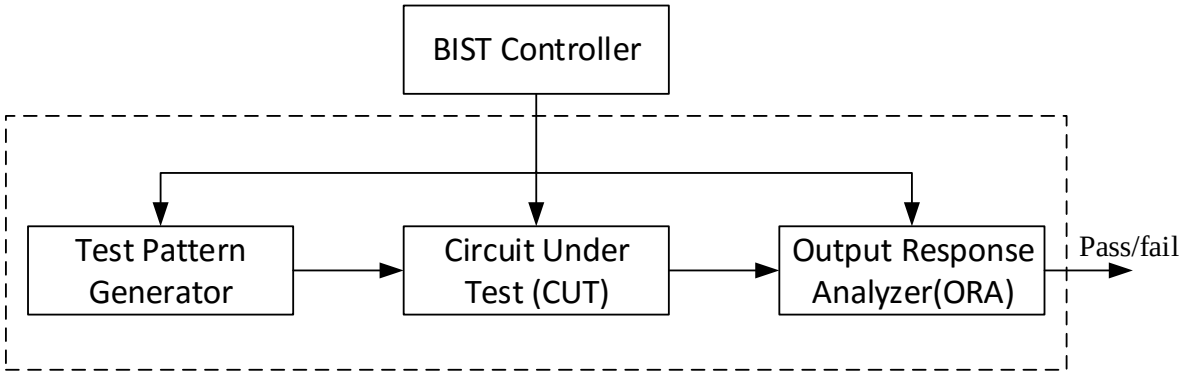


Figure 2.8: Basic BIST structure

To implement a BIST in a system, it requires additional effort and time to design the BIST circuitry and then to verify the device functionality both in the normal and test mode. Primarily, BIST costs in terms of area overhead which comes with the inclusion of BIST circuitry in the device. Similarly, performance penalties and power overhead during the test mode are also considered as significant disadvantages associated with BIST.

2.3.2.1 Types of test patterns in BIST

There are several types of test patterns that can be generated in the BIST environment. The fault coverage obtained in BIST mainly depends on the type of test patterns produced by test pattern generator. Brief explanation of some test patterns is given below:

Deterministic test patterns: These test patterns are used to detect specific faults in a given circuit. They are produced via ATPG or fault simulation. Usually these test patterns are stored in ROM or produced in a very specific way and they have limited applicability for BIST.

Algorithmic test patterns: these test patterns are developed to detect specific fault models in regular structures. They are typically repetitive, thus can be generated by finite state machine (FSM). These test patterns are mostly applied to BIST for RAM like structures.

Exhaustive test patterns: These test patterns include every possible combination of input test patterns. For *n*-input logic circuit, a counter can produce all 2^n test patterns which will be able to detect all detectable gate level stuck-at faults, wire AND/OR and dominant bridging faults. Exhaustive test patterns are not applicable for large *n*.

Pseudo-exhaustive test patterns: These test patterns exhaustively test each partitioned sub-circuit. *k*-input sub-circuit receives all 2^k possible test patterns where $k<n$. using Pseudo-exhaustive test patterns, all detectable gate level stuck-at faults, wire AND/OR and dominant bridging faults can be detected like exhaustive test patterns. But pseudo exhaustive is more adapted to large *n* as long as *k* is not large. These are commonly used in BIST applications. Counter, Linear Feedback Shift Registers (LFSRs) are usually used to produce pseudo exhaustive test patterns.

Random test patterns: As these test patterns are not repeatable, the fault coverage obtained after every execution will be different. Therefore, random test patterns are less likely to be used in BIST.

Pseudo-random test patterns: These test patterns are also random but their sequence is repeatable. They are also used in BIST applications and can be generated by LFSRs and Cellular Automata (CA).

2.3.2.2 Types of output response analyzers

An ORA produces a compacted signature of a Circuit Under Test (CUT) responses to the test patterns. That is to say, instead of observing outputs responses cycle after cycle, only the signature will be used to indicate the CUT as faulty/fault-free, at the end of all input sequences. There are several types of ORAs depending on the compaction of the CUT response that can be used in BIST.

Comparison-based ORA: This ORA compares the CUT response to test patterns and the fault free response stored in ROM to detect mismatches. The comparison is done for each test vector thus requiring a large amount of ROM to store the fault free response. For BIST, self comparator is more practical in which CUT output responses are compared with other CUTs' outputs rather than comparing with the stored fault free response.

Moreover, monitoring of every test vector response can be avoided by incorporating a latch to hold any mismatch in the previous comparisons.

Signature analysis ORA: This type of ORA is implemented using an LFSR. A signature is produced by dividing the polynomial representing the CUT output response and the polynomial of LFSR implementing the ORA. This signature is then compared to the signature of fault free circuit at the end of the BIST sequence.

Counter-based ORA: A counter is implemented in ORA which counts the number of '0s' or '1s' in the CUT output response. At the end of each test sequence, resultant count value is compared to the fault-free count value which indicates the faulty/fault free circuit.

In the following, we describe the implementation of BIST for FPGAs,

2.4 BIST for FPGAs

Intrinsic reconfigurability of an FPGA provides a very good opportunity for BIST implementation. BIST modules (TPG, ORA and BUT) are configured on the FPGA during test mode and then removed at the end of test completion. FPGA gets into normal application mode on the same resources, hence incorporating no additional hardware for TPG and ORA. BIST can be used for testing most of the FPGA resources, e.g. logic blocks, interconnect, memory cells, I/O blocks etc.

The first complete test and diagnosis method for FPGAs which formed the basis of modern BIST technique was proposed in [Stroud 1998; Abramovici 2001]. A single BIST operation is usually performed in two test sessions. In the first one, some CLBs in the FPGA are configured as TPGs and ORAs and some CLBs/SB as Blocks Under Test (BUTs). A BUT can be configured in a number of ways. Therefore, to perform an exhaustive testing, it requires that FPGA be configured in all possible modes of operation. For each configuration, a sequence of test vectors generated by TPGs are applied to BUT and the results are analyzed by ORA at the end of each test sequence. Thus, a set of test configurations is required to completely test a BUT targeting a specific fault model. At the completion of all test sets in the first session, the second session starts, in which CLBs swap their roles (TPG and ORA becomes BUT and vice versa) to complete the testing of the whole FPGA. A basic BIST structure for FPGA is shown in Figure 2.9. The number of test sessions required to test a particular FPGA may increase depending on the BIST algorithm developed for the FPGA targeting specific test time and fault coverage.

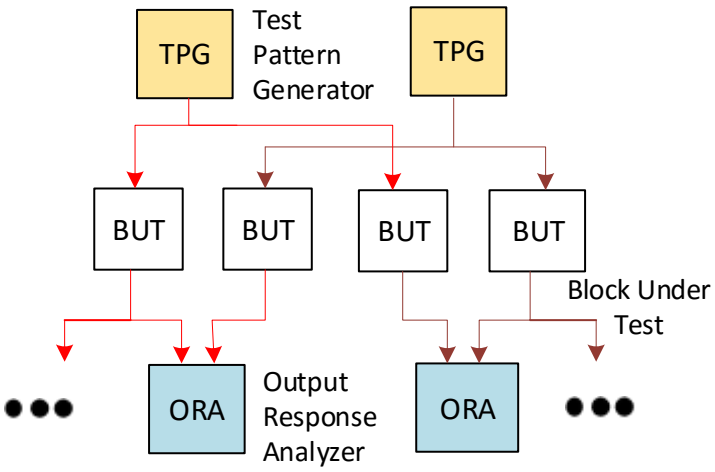


Figure 2.9: FPGA BIST structure

One of the prominent advantages of FPGA BIST is that it can be applied at various levels of testing which include, wafer-level, package-level, device-level, as well as system-level. Since BIST has become a mature technique over the past decade, high diagnostic resolution can be achieved which ultimately improves the device reliability. A CPU or special maintenance processors on board can be used as FPGA BIST controller which stores and manages the test configuration data. As no tester circuitry is required, area and performance penalties can easily be avoided. Furthermore, BIST can be performed at the inner device frequency because only the indigenous components are used.

BIST is an architecture specific technique. Its development and implementation depends on the FPGA architecture. Therefore any modification/optimization in the FPGA forces to develop a specific BIST scheme. In case of FPGAs, BIST configurations are developed based on the module under test. Usually, logic and interconnect resources are tested separately. The BUT undergoes a number of configurations during the test mode to guarantee the faults detection and diagnosis.

2.4.1 BIST application procedure in FPGAs

The application of BIST scheme can be explained in the following steps.

1) Reconfiguration of FPGA: For each BIST configuration, BIST bitstream stored in the BIST memory controller must be loaded on to the FPGA.

2) Execution of BIST sequence: It involves the initialization of BIST modules, followed by the generation and the execution of test sequences. Analysis of output responses is also done in this step.

3) Retrieval of BIST results: BIST results stored in the FF of the ORA logic blocks must be retrieved to determine the existence/non-existence of faults targeted in that BIST configuration.

In case of FPGA BIST, the main test cost is usually the test time. To determine the test time for a test session which consists of *N* number of configurations, time required to load every configurations, time required to execute and time required to retrieve the result from the flip-flop of the ORA is to be determined. Hence, the total test time T_{Test} for a given test session consisting *N* number of configurations is given by:

$$T_{Test} = \sum_{k=1}^{N} (T_{D,K} + T_{E,K} + T_{R,k}) \qquad \text{(eq. 2.1)}$$

Where $T_{D,k}$ is the time required to download the BIST configuration '*k*'. $T_{E,k}$ represents the time required for the execution of BIST sequence and $T_{R,k}$ is the time required to retrieve the results of the BIST sequence for the configuration '*k*'. Usually, BIST configurations are stored in the memory outside the chip on the FPGA board/system. For loading into the FPGA, these configuration bits are grouped into the frames of a certain bit-width (i.e. word) and are accessed one frame at a time and written into the SRAM cells. This procedure constitutes the reconfiguration time/download time $T_{D,k}$. The execution time involves the time required to produce test patterns and applying those patterns on to the BUTs. At the end of the test sequence execution, results are needed to be retrieved which are stored at the flip-flop of the ORA cluster. By using already embedded scan-chain mechanism or by memory read-back mechanism in most of the FPGAs the ORA flip-flops are read which constitutes the retrieval time.

The time to download a configuration dominates in the overall test time as the configuration time is considerably larger (approximately 100x) than the execution and the retrieval time. For a large FPGA, a full configuration may require 100ms to a few seconds depending on the clock frequency and configuration port width. A large number of test configurations is required for exhaustive testing with high diagnostic resolution. Therefore, the goal is to reduce the number of test configurations to reduce the overall test time.

2.4.2 BIST types for FPGA testing

BIST architectures for FPGAs have been studied extensively in the recent past years [Dutton 2009; Yao 2009; Zhu 2011; Kumar 2013; Almurib 2014; Tahoori 2003]. The motivations behind it include: reducing FPGA test time, ongoing improvement in FPGA architecture and exploitation of the FPGA features especially dynamic reconfigurations for testing and diagnosis purposes etc. FPGA BIST can be broadly classified in two categories: 1) offline BIST and 2) online BIST.

2.4.2.1 Offline BIST

In offline BIST, no application runs on the FPGA other than BIST configurations. BIST configurations are loaded into the FPGA one by one. All the FPGA resources are available to perform testing. Once testing is complete, BIST configurations are removed and the FPGA is reconfigured to the normal application to perform desired function.

In [Abramovici 2001], offline BIST technique for FPGA logic blocks was presented. It detects any single faulty CLB and any combination of multiple faulty CLBs. This technique relies on pseudo-exhaustive testing in which every module of CLB is exhaustively tested in each one of its mode of operation. The BIST structure is shown in Figure 2.10 which contains two TPGs producing identical test patterns. Each TPG feeds some specific BUTs. (i.e. Upper layer of BUTs are fed by one TPG and lower layer of BUTs are fed by other TPG). Then, an ORA compares the output responses of two BUTs that are fed by different TPGs. Since both TPGs produce identical test patterns, results of both BUTs should match in case of fault-free. If fault exist either in the BUT or in one of the TPG, the strategy of using two TPGs avoids the fault-masking phenomenon. It also eliminates the assumption that TPGs need to be fault free. However, this structure detects any combination of faulty BUTs as long as the two BUTs compared by the ORA do not fail at the same time, for the same cause. The contents of the ORAs are retrieved at the end of each test sequence by using scan-chain mechanism where the flip-flops of all ORAs are connected in series forming the scan-chain. The output sequence of the ORAs helps to locate the specific BUT in case of the presence of fault. This procedure constitutes one test session. To test the FPGA blocks which were configured as TPG or ORA in this test session, another session is needed in which FPGA blocks swap their roles i.e. TPG/ORAs are configured as BUTs and BUTs are configured as TPG or ORA. The floor plan for the first and second test session is shown in Figure 2.10 which shows that the blocks used as BUT in session one become TPG/ORA in session two and vice versa. If the number of logic blocks required to form the TPG/ORA of the BIST structure is larger than BUT, a whole BIST procedure is completed in two test session.

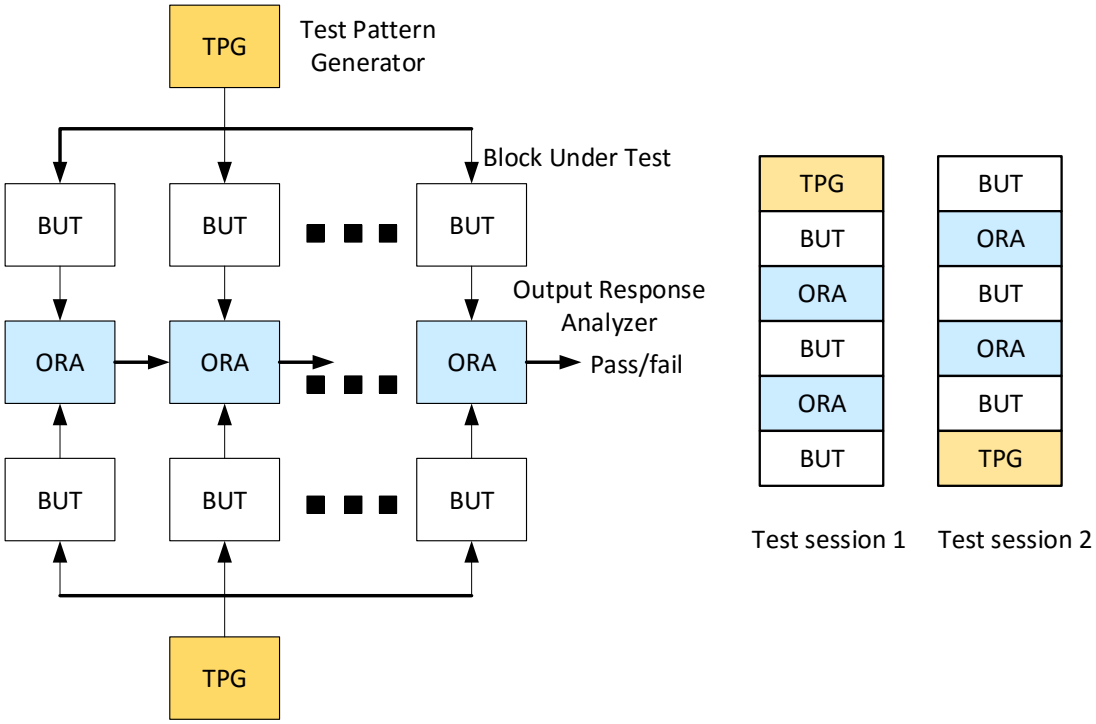


Figure 2.10: BIST structure and test sessions

Let us consider a scenario in which two faulty BUTs are analyzed in a common ORA having identical test patterns and identical faults. Since ORA is a comparison-based which finds a fault by detecting a mismatch between two BUTs outputs, the ORA will not be able detect fault in this situation and hence the fault will be masked and remain undetected. This discrepancy of the faulty BUT pair connected to a single ORA in [Abramovici 2001] is overcome by [Dutton 2009b] in which a circular BIST is used. In this structure, a given BUT is compared with two other different BUTs in two separate ORAs forming a circular connection between ORAs and BUTs. Figure 2.11 shows such architecture of a circular BIST. In this architecture, each BUT is compared twice in two different ORA of the same row whereas TPGs provide identical test patterns to the BUTs on alternate columns. In this way, each comparison in the ORA is done between two BUTs having identical test patterns but from different TPGs. As compared to the previous structure, the number of ORAs required in this structure is equal to the number of BUTs. Thus, circular BIST requires more than two sessions for a complete FPGA testing.

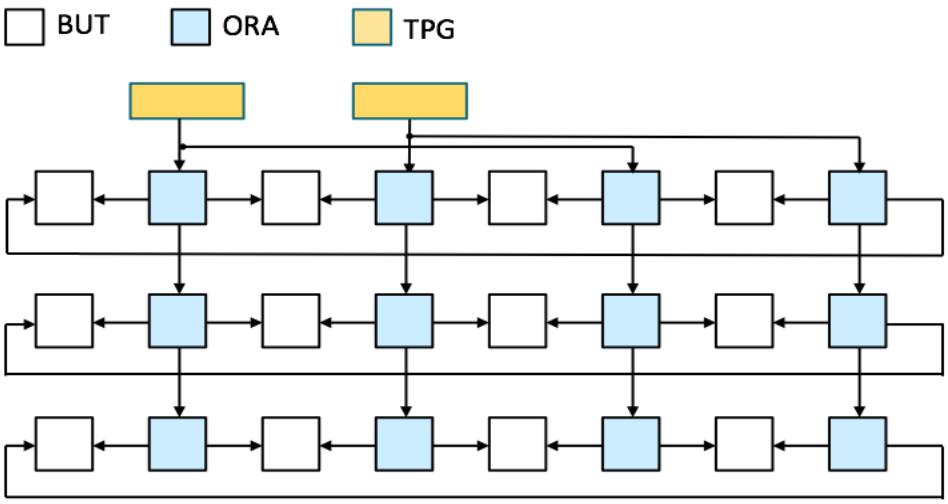


Figure 2.11: Circular BIST for FPGAs

In the same work [Abramovici 2001], a first complete testing of the configuration multiplexer was introduced which was either ignored or incorrectly solved in previous works [Huang 1996; Harris 2000]. A configuration is now commonly used in modern FPGAs; not only in logic blocks but also in interconnect structures as well. The configuration bits are used to control the configuration MUX by selecting one of its inputs to be connected to its output. Figure 2.12 shows a configuration MUX where a configuration bit '0' connects the input I_0 to the output O. In the normal mode, the value at the input I_1 and the sub-circuit producing it is ignored in since I_1 can no longer affect the output O. For the same reason, FPGA CAD tools do not include the inactive sub circuit while generating the configuration bitstream for a given application. But if a given application is for testing purposes, testing of the MUX test may not be completed. To completely test a MUX, all its unselected inputs are set to opposite logic values than the value of the selected input [Renovell 1997].

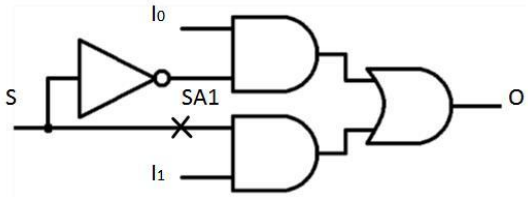


Figure 2.12: Configuration MUX

Figure 2.12 shows an example of a SA1 fault at the select input for I_1. To detect this fault, it is required to have S=0, 1_0=0, and I_1=1. To have a control value at I_1 while testing, the inactive sub-circuit needs to be configured such that it can generate proper logic value for the inactive MUX inputs.

Most of the pioneering work on FPGA interconnect BIST was done in [Huang 1996; Renovell 1998] which is based on externally applied test vectors and applicable only for device level manufacturing test. Advanced offline BIST for FPGA's global and local interconnect was addressed in [Stroud 1998] which only dealt with the fault detection and was unable to locate the fault. In this technique, a set of wire segments are selected by configuring cross points (configurable interconnect points in switch matrix) forming two groups of wire under test (WUTs). Identical test patterns are applied at WUTs and the responses are observed on the other side of WUTs. The WUTs may pass through CLBs (as in the case of intra-cluster interconnect testing). In this situation, CLBs are configured as a transparent block as shown in Figure 2.13.

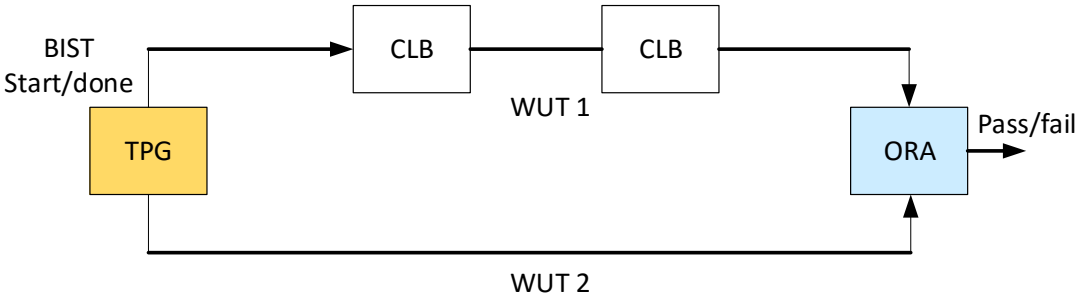


Figure 2.13: Interconnect BIST in FPGA

The same approach of interconnect BIST was extended in [Harris 2002] for the initial development of interconnect fault diagnosis in clustered FPGAs. In this work, a hierarchical technique was proposed based on the set of transparency constraints for controllability and observability perspective. On this basis, intra-cluster interconnect configurations are defined separately from extra-cluster interconnect configurations. This technique was developed for the FPGAs which are essentially composed of fully populated interconnect structures. Therefore, it is not compatible to the FPGAs having sparsely populated interconnect structures.

Similarly, [Yao 2009] presented a system-level BIST for global routing resources in Virtex-4 FPGAs. The developed BIST scheme is based on cross-coupled parity. Parity based BIST approach allows an odd number of WUTs making it easier to route WUTs. In cross-coupled parity BIST, a pair of TPG is configured in a cluster, one producing test patterns with even parity and the other with odd parity. Similarly, each TPG drives a pair of ORA where parities are cross-coupled. The goal is to increase the number of WUTs for any given test configuration and enhance the diagnostic resolution. However, the presented scheme is limited to Virtex™ FPGAs as it requires a large number of LUTs to be implemented.

2.4.2.1 Online BIST

Some FPGAs support run-time reconfiguration in which a portion of FPGA is reconfigured while the remainder continues to operate with minimal or no interruption. This very feature has been exploited to develop online testing and diagnosis of FGPA logic and interconnect resources. The first online BIST of FPGA logic resources was presented in [Abramovici 1999]. This technique proposed a roving Self Testing ARea (STAR) approach. A STAR is a temporarily offline/spare section of FPGA on which BIST modules (TPG/ORA) are configured.

In this on-line testing approach, parts of the FPGA say two rows and two columns and the associated routing resources are reserved for testing. The rest of the FPGA continues to perform the regular system functions. BIST structure is implemented on these two rows and two columns, thus these are referred to as STARs. A BIST floor plan of an FPGA containing an 8x8 array of CLBs is illustrated in Figure 2.14 in which vertical STARs (2 columns) and horizontal STARs (2 rows) are shown in dark along with the working area for system function shown in light colour. The routing resources in horizontal STAR connecting working areas above and below are ensured to remain in normal function to avoid any interruption in the working application. Similarly, routing resources in vertical STAR connecting working areas left and right are reserved for connecting the system partitions. The STARs are used for testing both the CLBs and the interconnect contained within the columns or rows of the STARs. Once the testing of logic and interconnect within the STARs has been completed, a portion of the system function running at the resources adjacent to STARs is relocated to the logic and interconnect resources within the STARs, thus creating new STARs on the new vacant resources. In this way, horizontal and vertical STARs rove back and forth across the FPGA, testing the logic and interconnect resources within the STARs [Stroud 2002].

Roving STAR approach can be used to detect both permanent and transient faults. It can be used to detect faults in logic block and memory resources as well. Moreover, [Abramovici 2003] extends the roving star to detect delay faults. Based on the roving STAR approach, [Dutt 2008] proposed new BISTer designs. These designs basically provided the number of TPGs and their connection with WUTs in FPGA array for improved diagnosis.

Online BIST is useful for the systems which require uninterrupted fault free operations. In the case of any fault occurrence in a system, normal operation can be replaced to fault free resources and thus more time can be allowed for accurate diagnosis of faults using online BIST. As compared to offline, online BIST requires more time to be

implemented. It is due to the reconfiguration time for operation relocation. Moreover, spare fault free resources are the basic requirement for an efficient online BIST operation. It is important to mention here that in this thesis we will focus on the development of schemes targeting offline test.

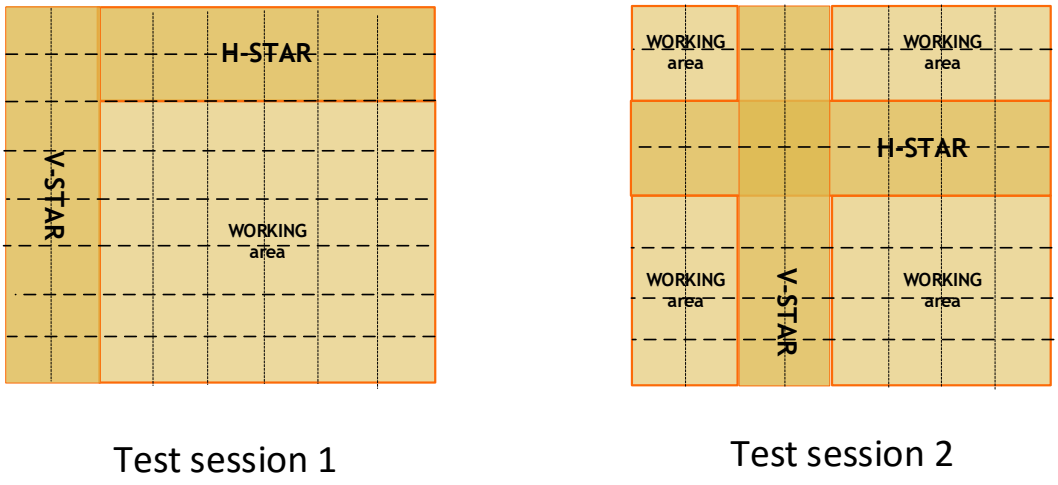


Figure 2.14: Online BIST in FPGA [Abramovici 1999]

2.4.3 Test time reduction

As mentioned earlier, the dominant component of the test time is the reconfiguration time which makes FPGA testing expensive. FPGA test cost can be reduced by minimizing the number of test configurations required to attain targeted fault coverage. In the past, several methods have been proposed to minimize the number of configurations needed to test an FPGA. Some require major modifications of FPGA's original architectures to make them self-configurable.

In [McCracken 2002], a scheme was presented in which switch matrices are modified to become self-reconfigurable. Test and diagnosis time is thus reduced by performing on the fly reprogramming to realize several test configurations with a single bitstream. This technique requires an addition of test structure in SRAM cells that controls the switch matrix. This structure includes a series of non-linear feedback shift registers added to each matrix. The new structure is claimed to reduce the switch matrix test time by 66% and diagnosis time by 72% at the cost of 4.5% area overhead.

An automated BIST architecture for test and diagnosis of FPGA interconnect faults is presented in [Smith 2006]. This approach requires BIST structures that contain self-enabling test pattern generators, self-configurable switch matrices, and response analyzers that can reprogram themselves without any external intervention. This eliminates the requirement of reconfiguration and hence reduces the test time.

Similarly in [Doumar 2000], FPGA's SRAM design is modified by implementing shift registers so that the new test configurations can be generated inside the chip by just loading a seed configuration. In [Zhu 2011] a cost-efficient interconnect BIST scheme is proposed. However, test time reduction is achieved at the cost of an area overhead by adding partially self-configurable structures. The additional self-configurable structures called test points are added only to the most efficient configuration port which is selected through analyzing test configurations. In this work, it is shown that test configurations for all interconnect stuck-at faults in Virtex™-II and Spartan™-3 FPGAs can be reduced by adding 1.2% area overhead. In [Fernandes 2003], authors developed a modeling and graph traversal algorithm using BIST to reduce the number of test configurations. In their work, the proposed scheme is applied to switch box (build connections between horizontal and vertical routing channels), without considering connection box (builds connections between logic and adjacent routing channels).

Some FPGAs support partial reconfiguration of their modules in which a portion of FPGA can be reconfigured without reconfiguring the remaining portions. This partial reconfiguration approach can be utilized for FPGA testing, to reduce the amount of test time and configuration bit storage cost. An online testing is presented in [Dutt 2008] where partial reconfiguration is used for testing spare resources without affecting the application running on the other part of FPGA. In this work, authors proposed 1- and 2-diagnosable BISTer designs based on a ROving TEster (ROTE) that moves across a functioning FPGA. It is claimed that the proposed 1-diagnosable functional-test-based BISTer has very high diagnostic coverage: e.g. for a random-fault distribution, the non-adaptive-diagnosis methods provide diagnostic coverage of 96% and 88% at fault densities of 10% and 25%, respectively. In [Legat 2010], an automated fault emulation approach is presented where SEU faults are injected during run time and only the resources affected by the faults are reconfigured using partial reconfiguration. BIST for Virtex and Spartan FPGAs using partial reconfiguration is given in [Dhingra 2005]. The proposed method utilizes only the differences between two consecutive BIST configurations thus reduces the total memory required to store the BIST configuration. Moreover, proper ordering of the sequence of BIST configurations gives speedy test which is more pronounced in the case of logic BIST that is found to be five times faster.

Most of the previous work on offline and online BIST dealt with commercial FPGAs mostly Xilinx and Altera. For the implementation and verification of such testing schemes, dedicated CAD tools were used which limit the benefit of the proposed schemes to the commercial FPGAs only. In addition to this, many BIST schemes test logic and interconnect resources separately and thus, do not exploit the architectural flexibility to

perform joint testing which could save test time. The schemes presented for test time reduction either propose major hardware modification in the basic architecture of the FPGA or require an extra effort for the generation of BIST configuration. Therefore, being proposed for specific architectures using specific tools, these schemes have limited compatibility with academic FPGA architectures providing the scope for the work done in this PhD thesis.

2.5 BIST design flow

BIST implementation requires the placement of the modules (i.e. TPG, ORA, BUT) and routing of the signals according to the developed scheme. However, the classic CAD tools used for configuration bitstream generation produce the default optimized placement and routing according to the algorithms running at their backend. Therefore, additional tools are developed to use classical CAD tools and conventional bitstream generation flow to implement the BIST scheme.

2.5.1 Xilinx FPGA flow

The CAD flow to produce the configuration bitstream in the case of Xilinx FPGAs is shown in Figure 2.15. The first step is the logic synthesis of the design description in Verilog or VHDL. After the synthesis, primitive gates are mapped onto the FPGA resource (i.e. logic cells, I/Os, specialized cores etc.) producing Native Circuit Description file. To intervene at the steps of placement and routing and make *.ncd* file humanly readable, Xilinx Design Language (XDL) file format is used. XDL was developed aiming at providing access to virtually all features of the Xilinx FPGA. For example, it provides a very powerful interface that can be used to constrain systems or directly implement modules or macros for Xilinx FPGAs. Moreover, XDL provides a human readable view of the FPGA resources and design netlist which makes it easier to translate and to verify test algorithms into the applicable constraints.

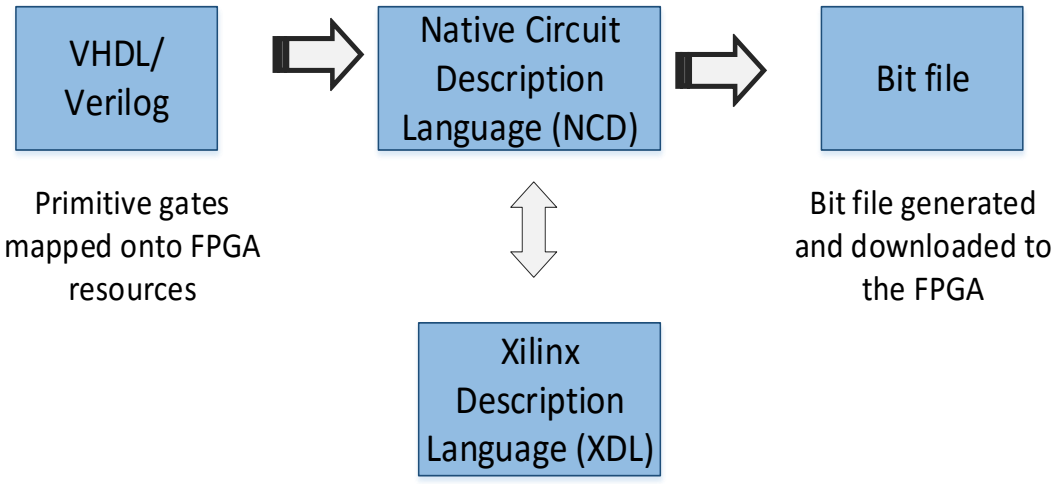


Figure 2.15: Xilinx bitstream generation flow

Using the basic flow for Xilinx FPGAs, bitstream generation flow for automated BIST is shown in Figure 2.16.

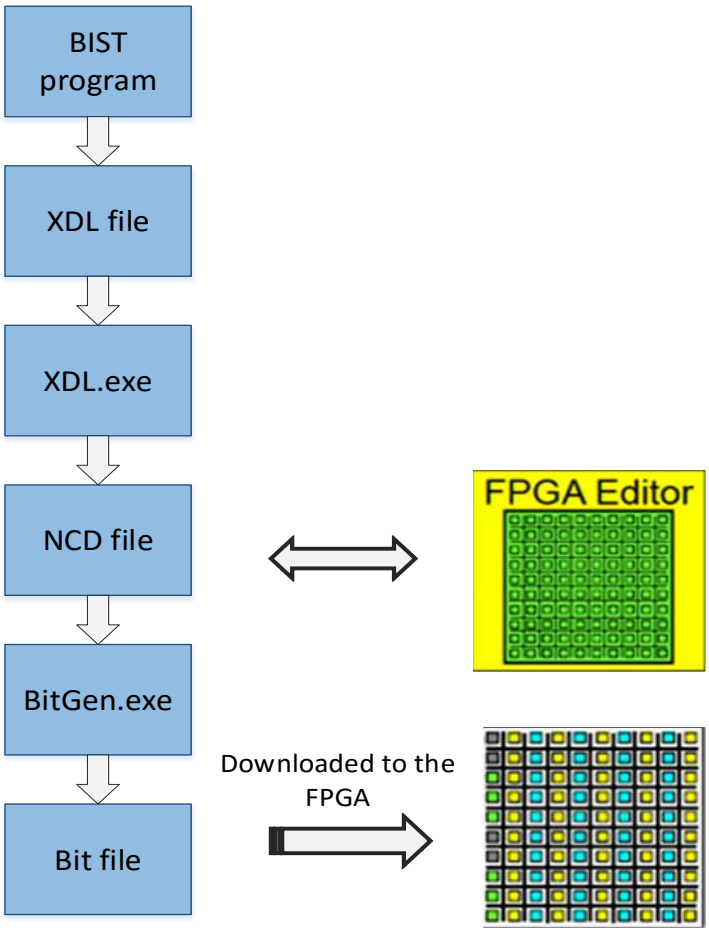


Figure 2.16: BIST for Xilinx FPGAs

The developers of BIST for Xilinx FPGAs [Dutton 2009] produced a lengthy BIST program (~500 lines of C code) defining placement and routing constraints as per their test algorithm. These programs thus generate the BIST template file in XDL format. XDL files are then converted into the NCD files using Xilinx indigenous program (xdl.exe). The NCD file represents the physical design mapped on to the components in the Xilinx FPGA. The internal place and route tools in Xilinx ISE take the mapped *.ncd* file, perform placement and routing of the design (test application in this case) and produce the NCD files that can be viewed in the FPGA editor. At the final step, Xilinx bitstream generation program (Bitgen.exe) takes the .NCD files as input and produces *.bit* file; the configuration bitstream that can be loaded onto the FPGA for testing.

2.5.2 JBits

It is evident that for any new test configuration or in the case of any modification in the test design, the complete configuration flow needs to be exercised again including synthesis placement and routing to get final bitstream. The amount of time required by executing again the flow to produce a number of test configurations is considerably large ranging from several minutes to hours [Dutton 2009]. This time cost can be reduced if configuration bitstream can be directly modified for the next test configuration once an initial bitstream is developed through the standard flow.

Xilinx offers JBits, the Xilinx bitstream interface to virtually access all configurable resources in the FPGA. JBITs is a set of Java™ classes which provide an Application Programmable Interface (API) into the Xilinx FPGA bitstream such that the bitstream can be read, modified or created by developing specific java applications. This interface operates on either bitstream generated by Xilinx design tools or on bitstreams read back from actual device [Guccione 2001].

JBits was actually developed to support run-time reconfiguration which allows modifying the circuit during the execution of the application on FPGA. Due to low-level design capabilities, JBits was later used to develop several tools including Virtex™ device simulator which directly emulates the FPGA hardware and accurately simulate the circuit behavior during reconfiguration.

Virtex™ devices support partial reconfiguration in which only modified configuration is loaded onto the FPGA keeping the other configuration unchanged. JRTR uses JBits API to identify and track the changes in the configuration data and only modified data is written to or read back from the device. For this reason, JBits has also been utilized for testing FPGAs. In [Sundararajan 2001; Niamat 2005], BIST scheme was presented for the detection and diagnosis of single and multiple stuck-at faults in CLBs.

BIST modules (i.e. TPG, ORA, BUT) were realized in terms of logic gates and JBits built-In methods were utilized to configure and to retrieve the data from memory resources. For example, *Jbits.set()* method was used to set the internal logic of the CLB as well as PIPs to a certain value i.e. on/off. *Jbits.read()* was used to read the input bitstream and to write a new bitstream *Jbits.write()* was used.

2.5.2.1 Advantages of JBits

The advantage of using JBits for testing FPGAs is that the test configuration bitstream are directly generated by Java codes eliminating the need to go through traditional lengthy configuration flow. In this way, the time required to generate configuration data can be reduced significantly. JBits can be used to build a database of the defects detected by tests and it has also the ability to configure around defects during the development of bitstream.

2.5.3 Limitations of commercial FPGA tools

All the features offered by JBits are limited to Virtex™ family of Xilinx FPGAs. Similarly, Xilinx ISE tool provides an efficient environment that can be used to implement BIST. As said before, it is only applicable to the Xilinx FPGAs, since the libraries and backend optimization processes are specific to the Xilinx FPGA architecture. For the same reasons, *Quartus* is dedicated for Altera FPGAs and Atmel utilizes *IDS* (Integrated Development System) software to place and route the designs. Therefore, *ISE, Quartus, IDS* cannot be utilized for the implementation of BIST for the FPGA in our case. To implement the BIST algorithms for a novel FPGA architecture, we need tools which provide freedom and flexibility to freely experiment with desired FPGA architecture and to implement the test schemes developed in this case.

2.6 Verilog-to-Routing (VTR) Project

VTR project developed by [Betz 1999], provides a complete design flow equivalent to Xilinx ISE for academic/hypothetical FPGA architectures. It involves open source tools in the flow which starts from a high level (i.e. HDL) circuit description, performs logic and physical synthesis, packing, placement and routing, down to the bitstream generation, allowing experimentation and optimization at any design level.

VTR consists of three core tools:

1) ODIN-II is a framework for Verilog HDL elaboration. It provides frontend synthesis and netlist flattering.
2) ABC is used for technology mapping and to optimize the soft logic of the FPGA.

3) VPR maps a technology mapped netlist to a hypothetical FPGA specified by the user.

In the end, *Bitgen* produces the bitstream that can be loaded onto the FPGA. Figure 2.17 shows a complete flow of VTR project to produce the FPGA configuration bitstream for a given FPGA architecture.

VTR Project serves the purpose of FPGA exploration mainly in two ways. 1) FPGA architecture development and 2) development in packing, placement and routing algorithms. For that reason, we decide to use *VTR Project* tools to develop our test and diagnosis schemes for the novel FPGA architecture. However, we need to develop a separate set of tools which can integrate our approach into the *VPR Project* flow for the implementation and verification purposes.

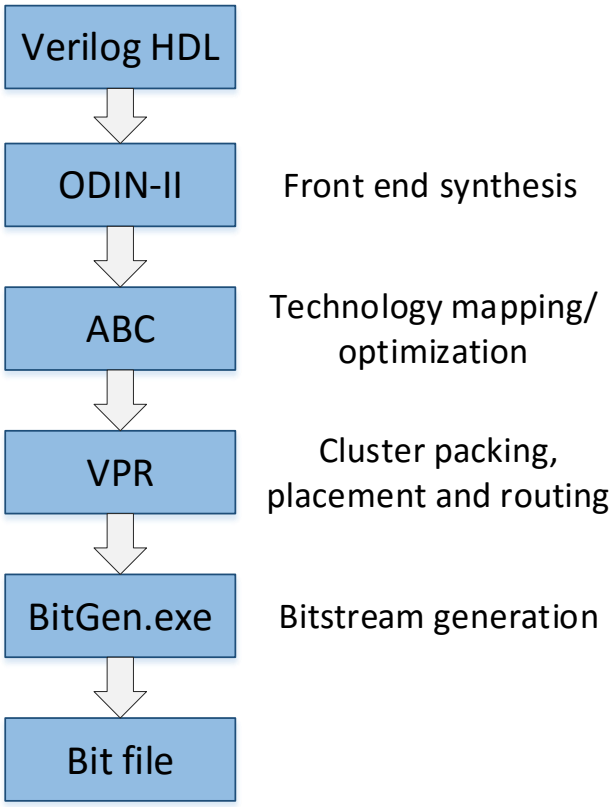


Figure 2.17: VTR Project flow for FPGAs

2.7 Conclusion

As we have seen that most of the previous work on FPGA testing deals with commercial FPGAs e.g. Xilinx and Altera. For testing such FPGAs, BIST has remained the preference by many researchers. It is due to the fact that BIST exploits the reconfigurable architectures well. BIST utilizes FPGA reconfigurability for its

implementation and does not incur extra hardware or performance penalties. At the completion of all test phases, normal application is run on the same resources. In addition to this, commercial FPGA CAD tools for configuration generation are used to implement BIST. Some advanced features and capabilities of these CAD tools can be utilized to improve the test time.

Although BIST is a generic technique, test configurations are architecture specific. For that reason, architecture customized configurations are needed to be developed for any new FPGA architecture. The FPGA architecture we consider in this thesis has a novel hierarchical interconnect which is based on sparsely crossbar structures unlike the commercial (e.g. Xilinx) or academic (*VTR-Project*) FPGAs. Therefore, it is imperative to develop efficient testing schemes exploiting the multilevel interconnect topology of such an FPGA architecture. Moreover, we need to develop tools to implement our proposed test schemes as well as to verify and evaluate their efficiency in terms of fault coverage and the number of required test configurations. For practical purposes, automation and generalization of the test schemes and their integration into the standard CAD flow are also needed. In the following chapter, the development of the new test schemes and the test time reduction methodologies will be discussed.

Chapter 3

Test and diagnosis schemes for novel FPGA

BIST approach is used for testing FPGAs for various reasons. The main advantages of BIST include 1) a reasonably high fault coverage with no area or performance overhead and 2) wafer to system level testing. FPGA BIST can detect both permanent and transient faults. However, BIST implementation requires the development of test configurations which depend on the given FPGA architecture. A new BIST scheme is defined to evaluate the testability and the test cost anytime FPGA architecture is modified. In this thesis, we consider a new FPGA architecture proposed in [Marrakchi 2010]. Further to that, we will assess the test methodologies developed for this new architecture.

3.1 Overview of the targeted FPGA architecture

Before discussing the test schemes, a necessary overview of the FPGA architecture under consideration is given below.

We consider a mesh FPGA in which several Configurable Logic Blocks (CLBs) are grouped together forming a cluster. Clusters are typically arranged in a grid surrounded by horizontal and vertical routing channels. These clusters are connected together through unidirectional interconnect network composed of wires and switch boxes as shown in Figure 3.1.

3.1.1 Cluster architecture

In a clustered FPGA, the number of CLBs in a cluster is known as the cluster size. In each cluster, CLBs are connected together using crossbar structures as shown in Figure 3.2. Depending on their connectivity, crossbars in the cluster are classified as down-linking (crossbar 'Down') and up-linking (crossbar 'Up') blocks. The crossbar 'Down' offers the flexibility of connecting limited number of cluster inputs to a given CLB. The crossbar 'Up' connects CLBs' outputs to the cluster outputs as well as provides local

feedbacks to CLBs through the crossbars 'Down'. It is important to note that these feedback paths are also sparsely distributed among crossbars 'Down'. The crossbars 'Down' in the cluster are sparsely populated meaning that the cluster inputs are uniformly distributed among these crossbars.

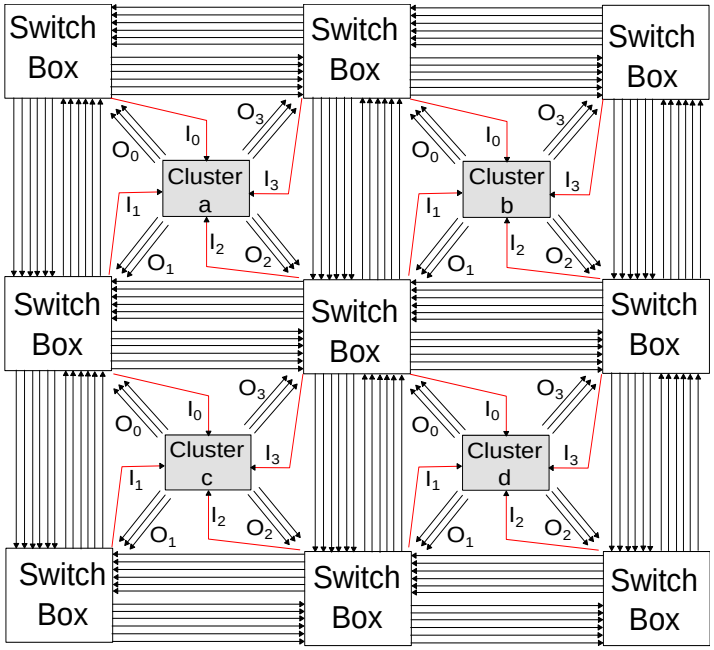


Figure 3.1: Mesh of clusters FPGA

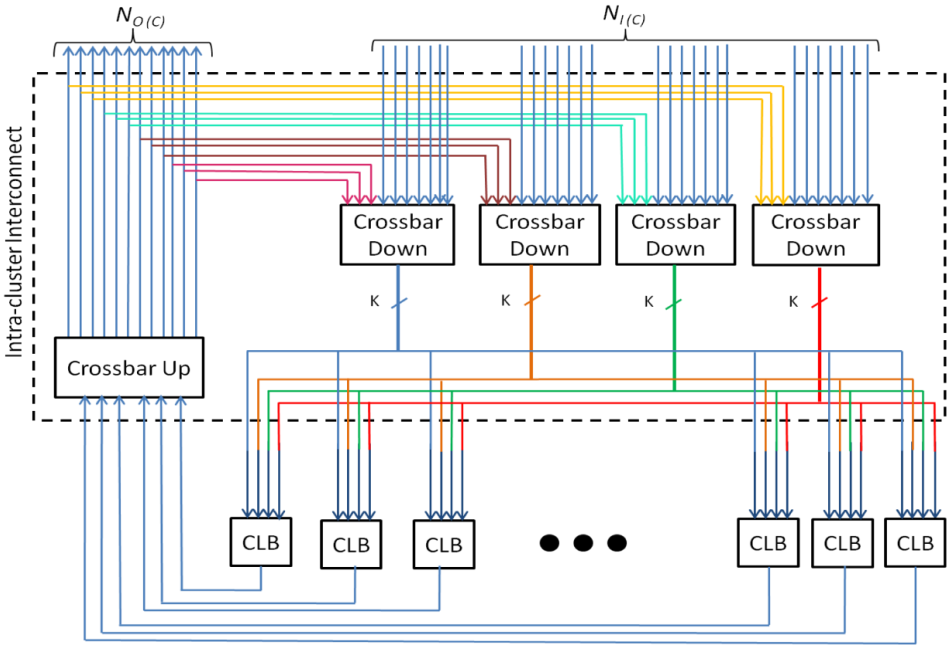


Figure 3.2: Cluster in a mesh FPGA with depopulated crossbars

The 'Up' and 'Down' crossbars are multiplexer based structures and an example of a crossbar 'Down' in the cluster of size 12 is shown in Figure 3.3.

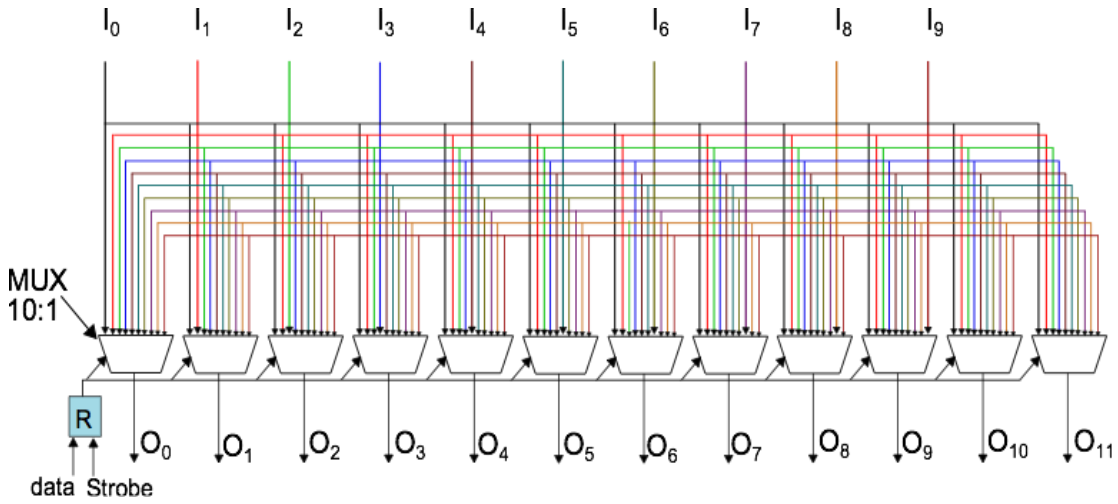
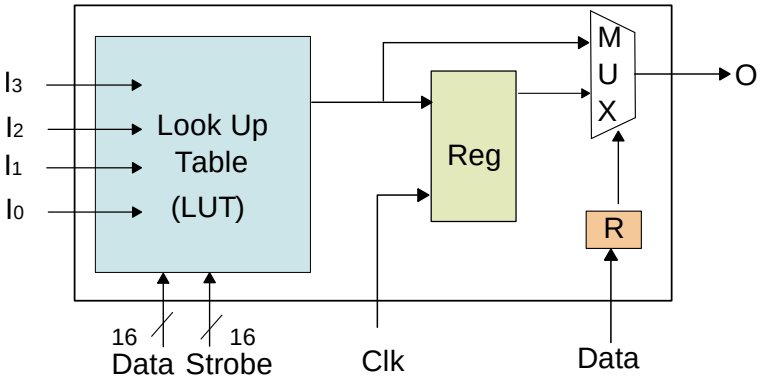


Figure 3.3: Structure of a crossbar

3.1.2 CLB architecture

Each CLB in a cluster contains a single LUT, a register and a multiplexer. 4 inputs LUT (LUT-4) is considered as the most area-efficient and therefore used in most of the modern FPGAs [Ahmed 2004]. Figure 3.4 (a) represents a CLB with a 4-input LUT (LUT-4), a Flip-Flop and a multiplexer 2:1 (MUX2) to select either the combinational or the sequential path. A conventional LUT-4 is depicted in Figure 3.4 (b). It is based on SRAM cells followed by a series of MUX2s. The configuration bitstream is loaded in the SRAM, while the MUX2s implement the logic function, according to the values of inputs A, B, C and D.



a) Configurable logic block in the cluster of a mesh FPGA

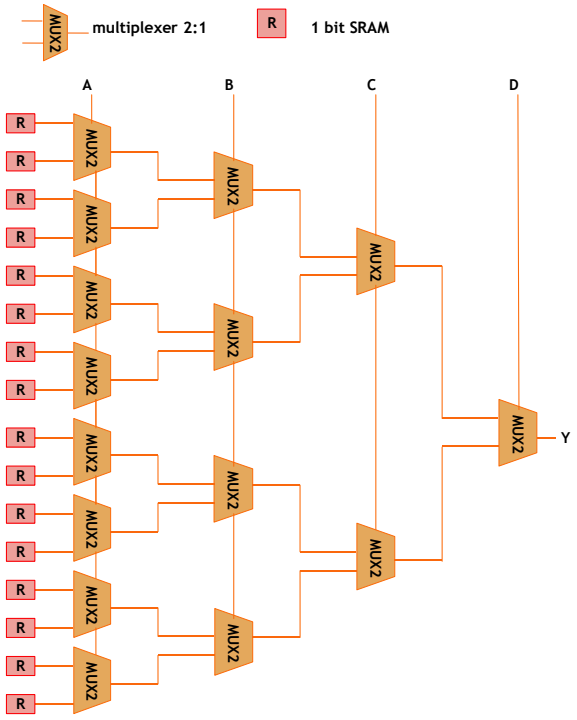


Figure 3.4: a) CLB and b) 4-input LUT in the cluster of a mesh FPGA

3.1.3 Cluster size optimization

The amount of logic functions implemented in a cluster is defined by the number of LUTs/CLBs per cluster. Therefore, higher cluster size is preferred as the length of critical paths is reduced (more logic being inside a cluster), thus improving the performance. However, the increase in cluster size is limited by the increase of cluster interconnect (local interconnect) complexity and area which grows quadratically with the number of CLBs per cluster [Ahmed 2004]. Therefore, a trade-off is usually explored between the number of cluster input/output and the size, with the help of Rent's rule. According to this rule,

$$N_{I/O} = KxC^{r} \qquad \text{(eq. 3.1)}$$

Where $N_{I/O}$ is the number of input/output pins of the cluster, K is the number of LUT I/Os, C is the cluster size expressed in terms of number of CLBs per cluster and r is the Rent's parameter which gives a common measure for interconnect complexity and typically $0.7 < r < 1$.

For the FPGA architecture under consideration, $r=0.84$ is found to give maximum utilization of the LUTs [Amouri 2013]. Architecture of each cluster size considered in

this experiment is developed in agreement with the Rent parameter. According to this rule, the number of cluster input and output pins for different sizes is given in Table 3.1. The number of inputs and outputs of each cluster must be a multiple of 4 for a symmetrical tile structure of the FPGA mesh. We consider the cluster of size 4, 6, 8, 10 and 12 for implementation procedures in this work. Higher cluster sizes are not considered due to the architectural limitation of the FPGA [Marrakchi 2009]. Moreover, a 4-input LUT has been used for all the cluster sizes.

TABLE 3.1 CLUSTER I/OS FOR DIFFERENT SIZES

Cluster Size	Number of Cluster Inputs	Number of Cluster Outputs
4	12	4
6	16	8
8	20	8
10	24	12
12	28	12

3.1.4 Switch box architecture

In a mesh FPGA, the clusters are connected together through unidirectional interconnect network composed of wires and switch boxes (cf. Figure 3.1). Along with the connection among clusters, the switch boxes also provide connection between horizontal and vertical routing channels which are essentially based on unidirectional wires. In this FPGA architecture, the switch box is made of multiplexer-based sparsely populated crossbars similar to those in the cluster.

Inside, the switch box connections are composed of two hierarchical levels. The 'Level 1' relates to connection among adjacent clusters whereas 'Level 2' connects adjacent switch boxes together. Figure 3.5 shows the internal structure of a central switch box in a mesh FPGA (cf. Figure 3.1), connected to other four adjacent switch boxes and to four clusters. The crossbars used in the switch box are sparsely populated meaning that the inputs of the switch box are uniformly distributed among the crossbars. Sparsely populated crossbar offers considerable area saving without loss of much routability as compared to fully populated crossbar [Marrakchi 2009]. Being unidirectional in nature, crossbars in the switch box are further classified as downward and upward linking blocks. In this FPGA architecture, these blocks are named as Downward Mini Switch Box

(DMSB) and Upward Mini Switch Box (UMSB) respectively. An example of a mini switch box with 4 inputs and 3 outputs is shown in Figure 3.6.

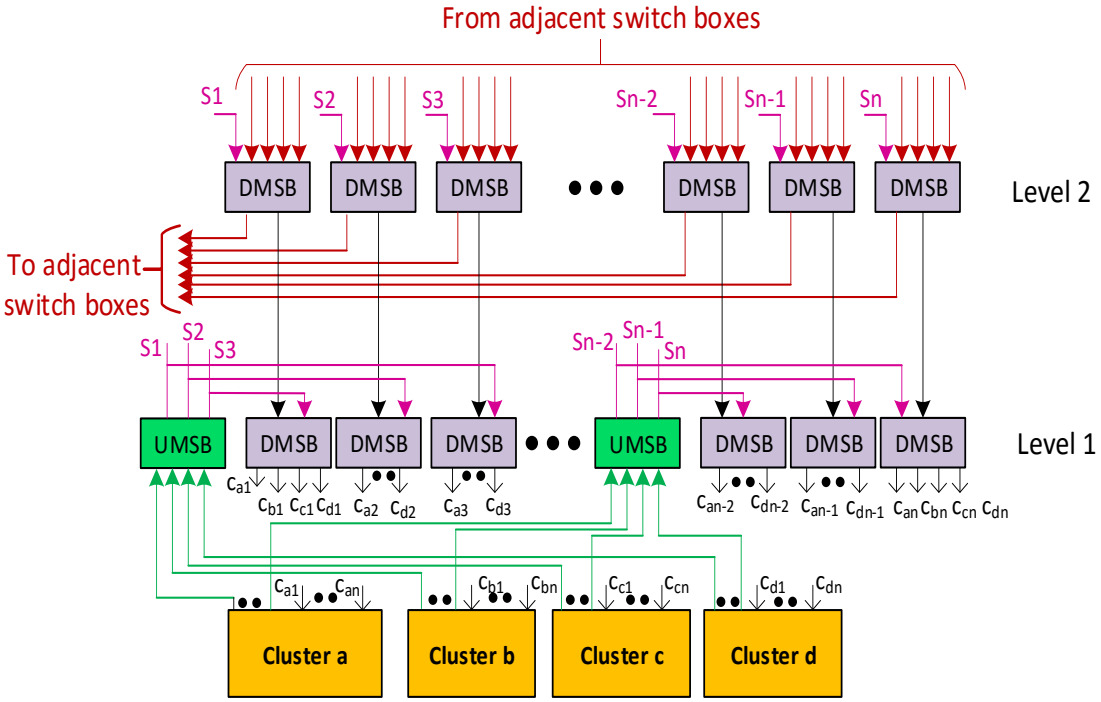


Figure 3.5: Structure of a Switch Box in a mesh FPGA

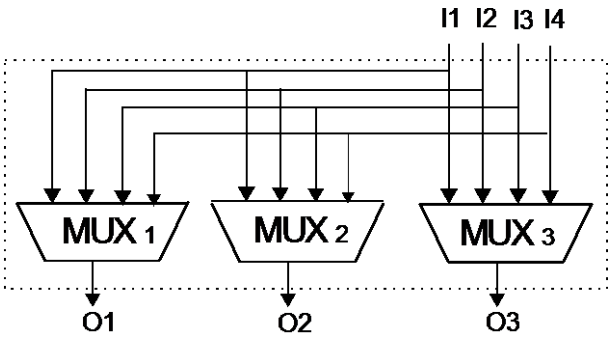


Figure 3.6: Example of a mini switch box with 4 inputs and 3 outputs

The Upward Mini Switch Box (UMSB) connects outputs of adjacent clusters to the switch box. In practice, more than one UMSB are used to keep the UMSB sparsely populated. The number of UMSBs in a switch box depends on the number of adjacent cluster outputs. The outputs of each adjacent cluster are uniformly distributed among all UMSBs such that each UMSB is connected to a specific number of outputs coming from each cluster. The UMSB lies at 'Level 1' of the interconnect hierarchy (see Figure 3.5) and connects the clusters outputs to DMSBs used in 'Level 1' and 'Level 2'.

The Downward Mini Switch Box (DMSB) connects the switch box inputs to the inputs of adjacent switch boxes and to the inputs of adjacent clusters (Ca1, …Can, Cb1, …Cbn, Cc1, …Ccn, Cd1, …Cdn). The outputs of DMSBs in 'Level 1' of the switch box are connected to the inputs of adjacent clusters. The number of DMSBs at 'Level 1' depends on the number of inputs of adjacent clusters (here we have considered four adjacent cluster, switch box being at the centre of the mesh) as well as on the condition of connecting a unique output of each cluster to the input of any adjacent cluster via UMSB. The DMSBs at 'Level 2' connect adjacent switch boxes together. Each DMSB at this level ensures its connectivity to each adjacent switch box both at its inputs and outputs. Therefore, the number of inputs/outputs of a DMSB at 'Level 2' corresponds to the number of adjacent switch boxes.

3.2 Test and Diagnosis Methodology

As discussed in chapter 2, one BIST operation is usually completed in two test sessions. In the first session, some groups of CLBs in the FPGA are configured as Test Pattern Generators (TPGs) and as Output Response Analyzers (ORAs) and some others as Blocks Under Test (BUTs). As BUTs can be configured in a number of ways, several test configurations are required to completely test a BUT. For each configuration, specific test patterns generated by TPG are applied to the BUT and the results are analyzed by the ORA at the end of each test sequence. At the completion of all test configurations, a second test session starts in which CLBs swap their roles (TPG and/or ORA becomes BUT and vice versa). The objective of these two test sessions is to configure every FPGA block as a BUT at least once to complete the test of whole FPGA.

Similar to the conventional BIST approach for FPGAs, test and diagnosis is performed here on modular basis. It means that the logic and interconnect resources are tested separately.

Therefore, we divide our test and diagnosis methodology in two phases.

1) In phase 1, CLBs and crossbars in the cluster are tested and,
2) In phase 2, the switch boxes are tested.

The fault models that are targeted during these test sessions include single stuck-at and pair-wise bridging fault models. In all the test phases and sessions, basic functions configured for TPG and ORA remain unchanged. TPG and ORA functionality is explained as follows.

It is important to mention that our proposed methodology performs both the faults detection and diagnosis. The term 'test' is used hereafter for diagnosis purposes referring to both fault detection and location.

3.2.1 Test Pattern Generator (TPG)

We need pseudo-exhaustive test patterns to test the CLBs and interconnect. These types of test patterns guarantee the detection of all detectable gate-level stuck-at and bridging faults. Pseudo-exhaustive test patterns can be generated using n-bit LFSR or n-bit counter where 'n' is defined by the number of Paths Under Test (PUTs). To implement an LFSR or a counter in the cluster, CLBs are configured to implement the respective function.

For example, we use type-2 LFSR in which flip-flops in the CLBs are used as shift registers and XOR function is implemented on the LUT of the CLBs. To produce the 4-bit test patterns for testing the LUT-4, 4 CLBs are required where we implement the following polynomial.

$$P(x) = x^4 + x^3 + 1 \qquad \text{(eq. A)}$$

All 0s patterns are avoided as an LFSR gets stuck in this state without a proper solution to deal with this configuration.

Taking advantage of the large cluster-size and small LUT-size in the FPGA architecture, more than one TPG can be implemented in one cluster. Using multiple TPGs produces identical test patterns thus eliminating the possibility of fault masking due to faulty TPG.

3.2.2 Output Response Analyzer (ORA)

To analyze the outputs of the BUT clusters, we use a comparator-based analyzer. For that purpose, we configure the CLBs in a cluster (which we call ORA cluster) as a 3-bit comparator. This comparison occurs after each test vector of a test sequence has been applied on the circuit and the output has been produced. In order to store mismatch for any test vector till the end of test sequence, we use the feedback path available in the cluster through crossbar 'Up'. In this way, we can compare the 2 outputs from the BUT cluster. These outputs could be from a same BUT cluster or different. In order to avoid fault masking, it is preferred to compare outputs coming from different BUTs which have been fed with different TPGs.

In an ORA cluster, each CLB can be configured as an independent comparator which compares the two outputs each from different BUTs. Figure 3.7 shows such a CLB comparator implemented in an ORA cluster where two CLBs from two BUTs are compared. $CLB_{1\ BUT1}$ and $CLB_{1\ BUT2}$ are the outputs of BUT_1 and BUT_2 respectively. FB_{1ORA} is the feedback which is used to store the previous mismatch. It is critical to configure the crossbar 'Up' and crossbar 'Down' in the ORA cluster to connect the FB_{1ORA} signal to the proper CLB. As mentioned before, this type of ORA does not require fault free response for comparison, thus saves the memory cost.

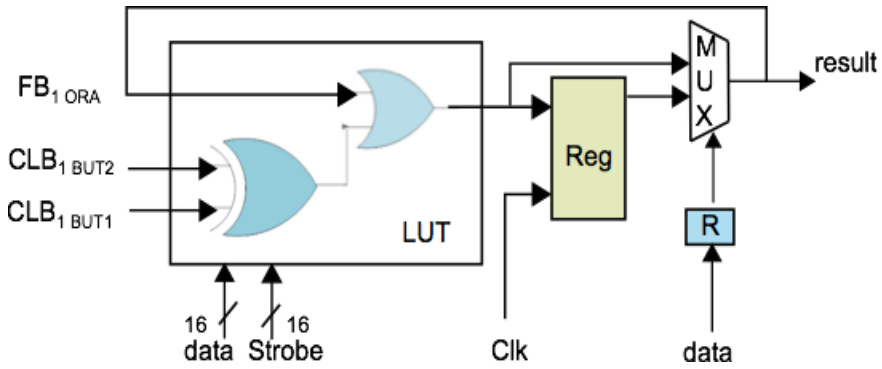


Figure 3.7: Logic block of comparison-based ORA cluster

In the following, we will discuss the BIST structure and test configurations for testing the CLB and the crossbars in the clusters of the FPGA.

3.3 Test methodology for CLBs

The main module to be tested in the CLB is the LUT which is based on SRAM cells followed by a series of MUX2s. The LUT should be tested in all its modes of operation. There are two main modes of LUT operation. 1) RAM mode of operation 2) LUT mode of operation. In RAM mode of operation, faults in the SRAM cells of the LUT are detected. For that purpose, TPGs can be configured to apply a March test to detect all stuck faults in the memory cells as well as all the faults in address and read/write circuitry. Other fault models can be tested as well based on classic March testing for memories. However, in this thesis, we assume that all the SRAM cells in the FPGA are fault free. Therefore, LUTs are not tested in their RAM mode of operation.

In the LUT mode, LUT is tested for stuck-at faults at every input of its multiplexers i.e. the configuration inputs coming from SRAM cells (normal inputs) as well as inputs from the crossbar 'Down' (select input). For a complete testing, each input is selected at least once and only one input (among 2 normal and a select input) is changed which changes the MUX output.

This is equivalent to XOR-function which identifies any change at its inputs. For that reason, two configurations are used to test the LUT-4.

1) In the first one, LUT is configured as XOR function and a 4-bit test sequence is applied at its inputs. A complete test of a MUX requires all its unselected inputs to have an opposite logic value than that of a selected input. The output response is then analyzed to detect any stuck-at faults in any of its inputs.
2) Similarly, in the next configuration, LUT is configured to have XNOR functionality to detect any stuck-at faults that may not be covered in the first configuration.

To implement a BIST architecture, TPG, ORA and BUT are connected with the help of routing netlist which is generated according to the BIST scheme. An exemplary scenario is depicted in Figure 3.8 in which connections among four BIST clusters are shown. TPG Cluster 'a' is configured as an LFSR, producing identical test patterns for each CLB of BUT clusters 'b' and 'c'. Each CLB of cluster 'd' is a comparator based ORA i.e. $CLB_{1\ ORA}$ compares $CLB_{1\ BUT\ b}$ and $CLB_{1\ BUTc}$. For the sake of simplicity, crossbars and switch box connections are not shown.

Two additional configurations are required to test the registered LUT output path to ensure a complete CLB testing. From the cluster architecture, we know that, CLB inputs and outputs can be accessed only through crossbar 'Down' and crossbar 'Up' respectively. To do so, each crossbar 'Down' is set to an appropriate configuration to provide test patterns to each CLB. Similarly, crossbar 'Up' is configured such that each CLB can be observed at the cluster output.

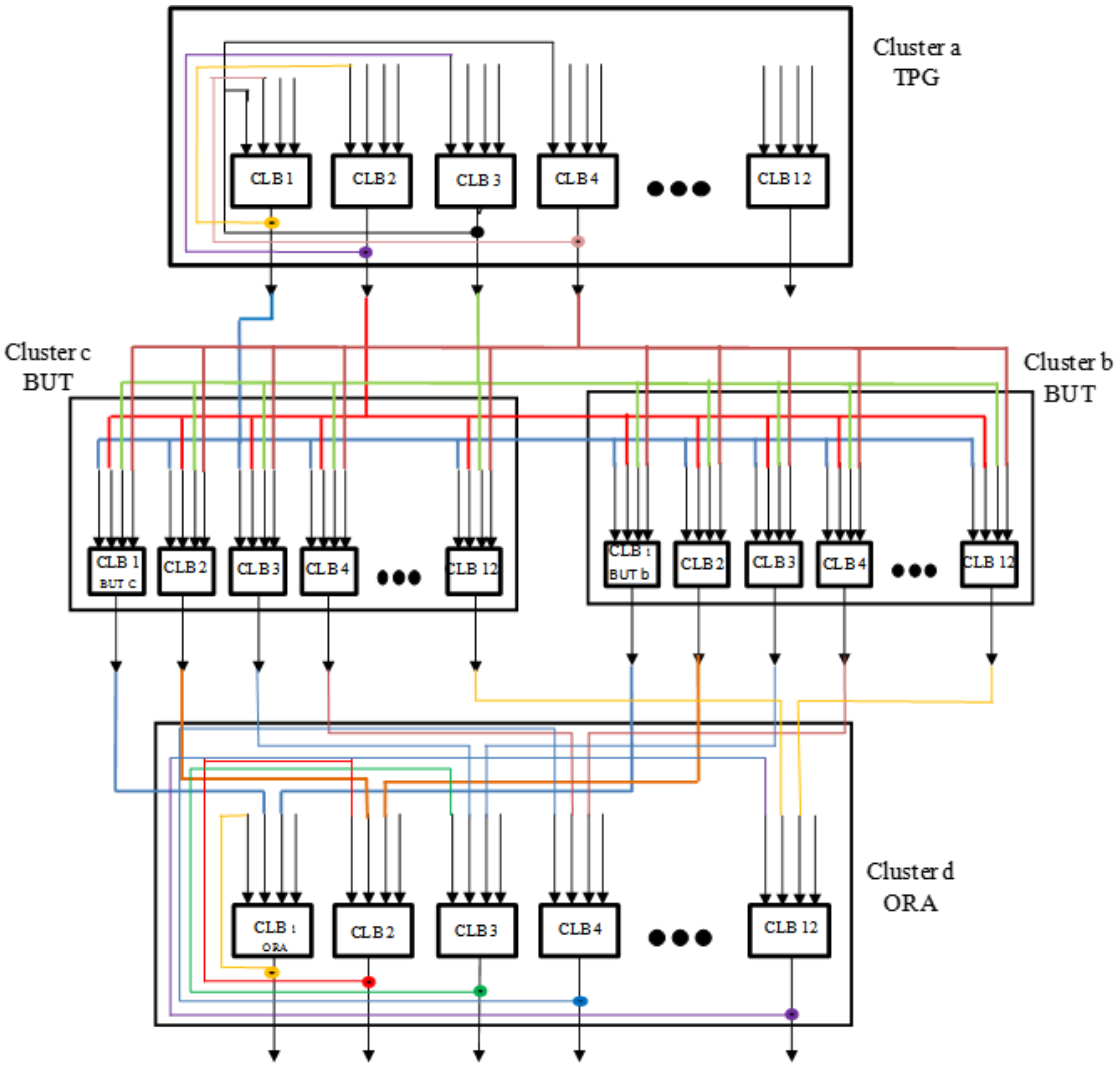


Figure 3.8: Exemplary scenario of BIST

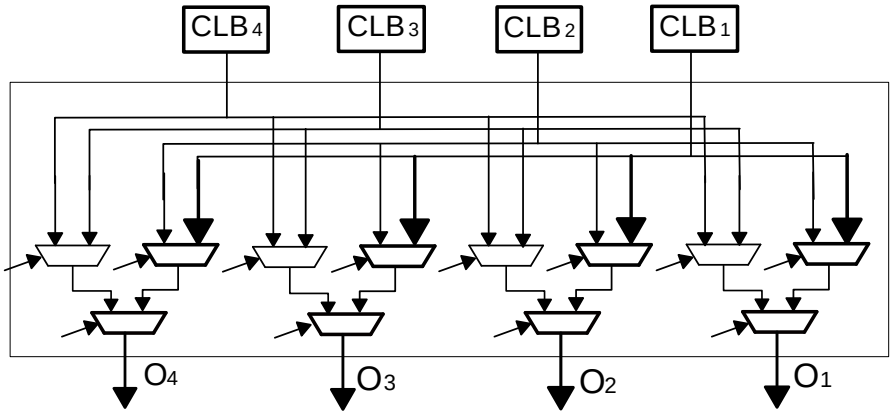
3.4 Test methodology for crossbar 'Up'

The crossbar 'Up' is a multiplexer based structure. The number of cluster outputs defines the number of MUXes in the crossbar 'Up' whereas the number of CLBs in the cluster defines the MUX size. For a large cluster size, large MUXes are used in the crossbar 'Up'. To completely test a MUX, all its inputs are selected one by one; hence a large number of configurations are required for testing a large MUX.

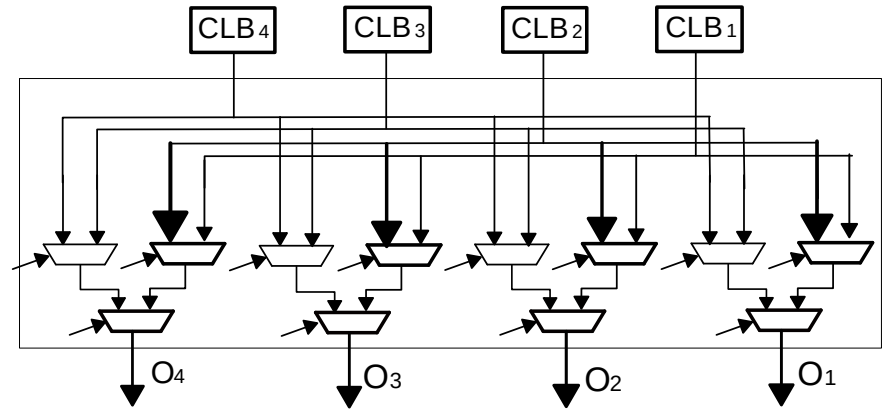
As mentioned earlier, a complete testing of a MUX requires all its unselected inputs to have an opposite logic value than that of a selected input. Therefore, for testing MUXes in the crossbar 'Up', the CLBs and crossbar 'Down' need to be configured to fulfill this condition as the test patterns are propagated through them in the BUT cluster. We consider a crossbar 'Up' in a cluster of size 4 as an example shown in Figure 3.9. In

this example, four 4:1 MUXes are shown where each MUX is made of 2 hierarchical levels of 2:1 MUXes. Since the outputs of all four 4:1 MUXes are connected to the cluster outputs, they can be tested in parallel. Figure 3.9 a) shows the first configuration, where the path under test is shown in bold. To have the proper test pattern for this configuration, CLBs are configured as transparent blocks. Only one input of the CLBs among 4 containing a test bit is propagated through. The LUT configurations to propagate its inputs A, B, C and D are shown in Table 3.2.

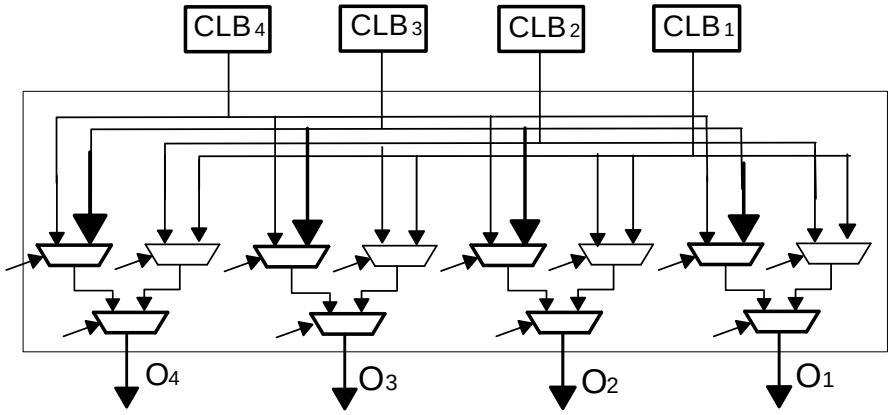
Figure 3.9 b) shows the second input of the crossbar 'Up' being selected for test. In this way, it needs 4 configurations to completely test the crossbar 'Up' of a cluster with size 4. As the cluster size increases, the number and size of MUXes in the cluster increases as well. As a result, the hierarchical level made of 2:1 MUXes increases. To completely test such large MUXes, the number of required test configurations does not remain linear with the cluster size. The number of test configurations required to test the crossbar up with respect to different cluster sizes will be explained in chapter 6. All the way throughout the testing of the crossbar 'Up', the configurations of crossbar 'Down' and CLBs remain unchanged.



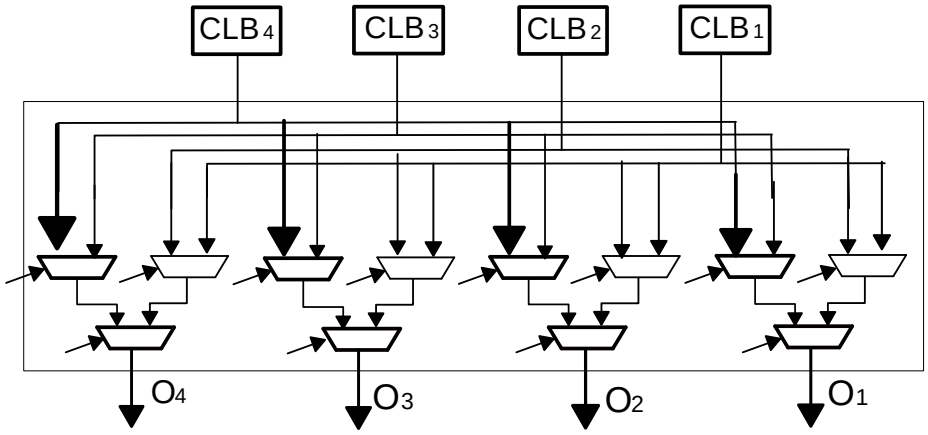
a) First configuration for testing first input of each MUX in crossbar up



b) Second configuration for testing second input of each MUX in crossbar up



c) Third configuration for testing third input of each MUX in crossbar up



d) Fourth configuration for testing fourth input of each MUX in crossbar up

Figure 3.9: Sequence of test configurations for crossbar-up in cluster-size 4

TABLE 3.2 LUT CONFIGURATIONS TO PROPAGATE AN INPUT TO OUTPUT

The LUT SRAM configuration	Propagating A	Propagating B	Propagating C	Propagating D
SRAM[0]	0	0	0	0
SRAM[1]	1	0	0	0
SRAM[2]	0	1	0	0
SRAM[3]	1	1	0	0
SRAM[4]	0	0	1	0
SRAM[5]	1	0	1	0
SRAM[6]	0	1	1	0
SRAM[7]	1	1	1	0
SRAM[8]	0	0	0	1
SRAM[9]	1	0	0	1
SRAM[10]	0	1	0	1
SRAM[11]	1	1	0	1
SRAM[12]	0	0	1	1
SRAM[13]	1	0	1	1
SRAM[14]	0	1	1	1
SRAM[15]	1	1	1	1

3.5 Test methodology for crossbar 'Down'

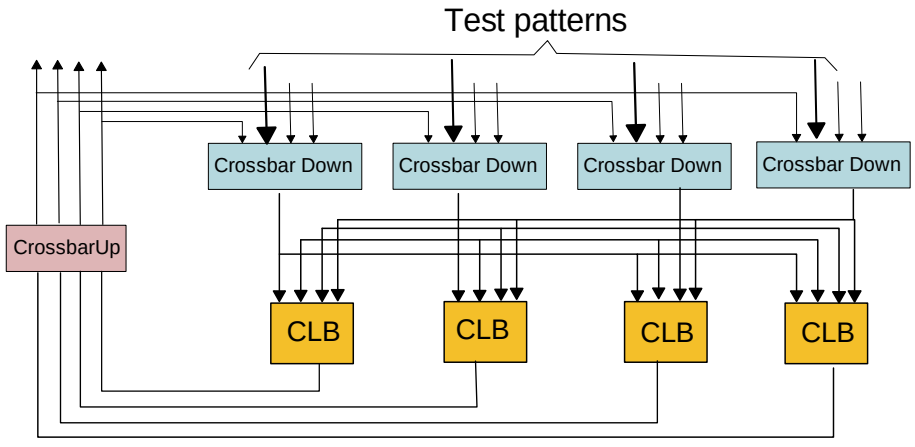
To perform crossbar 'Down' testing, an approach similar to the crossbars 'Up' is followed, i.e. parallel testing of all four crossbars 'Down' in the cluster. The number of MUXes in each crossbar 'Down' is equal to the cluster size. For an exhaustive testing of all MUXes in each crossbar 'Down', one input of each crossbar 'Down' will be tested at a time. From the cluster architecture, we have seen that each crossbar 'Down' is connected to only one CLB input. In this way, applying 4-bit test pattern to activate four Paths Under Test (PUTs), (one through each crossbar 'Down'); each CLB receives identical test pattern at its inputs. The CBU is configured such that each CLB output can be observed right at the cluster output. In this way, four PUTs from four crossbars 'Down' are

converged to one PUT in the CLB as CLB has only one output. If each CLB is configured as XOR/XNOR and all PUTs are sensitized with identical test patterns (all 0s or 1s), any mismatch among these four PUTs can be identified at the cluster output. This is shown in Figure 3.10 where the first input of each crossbar 'Down' is tested in the first configuration. (PUTs are shown in bold). All 0s test pattern is applied and CLBs are configured as XORs. Similarly, the second configuration tests the second input and so on.

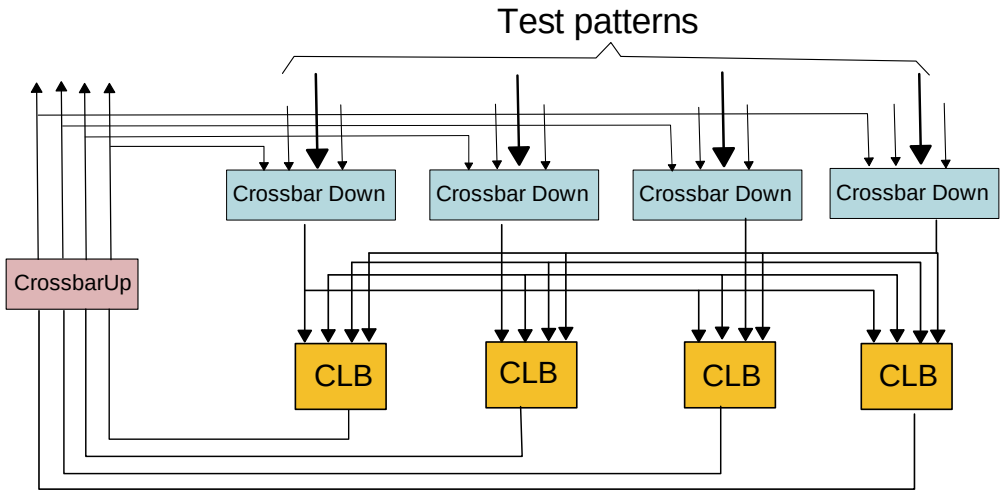
In order to test the crossbar 'Down' inputs coming from crossbar 'Up' (i.e. feedback paths), crossbars 'Down' are reconfigured to select these feedback inputs after first selecting the non-feedback input for the test patterns. Since, the cluster output is feedback as the test pattern for these feedback paths, any mismatch between consecutive configurations in the cluster outputs indicates the faulty output.

This configuration scheme of testing crossbar 'Down' can detect faults among similar PUTs from four crossbars 'Down'. However, it is unable to locate the exact faulty crossbar 'Down' or MUX. The reason is the following: the connection made among CLB and crossbar 'Down' make impossible to identify which of its input is faulty. Moreover, this configuration scheme can also mask the faults if any pair of two or all four PUTs are faulty. Although, parallel testing of all four crossbars 'Down' is possible for fault detection, the 100% diagnosis resolution of faults at MUX level cannot be achieved.

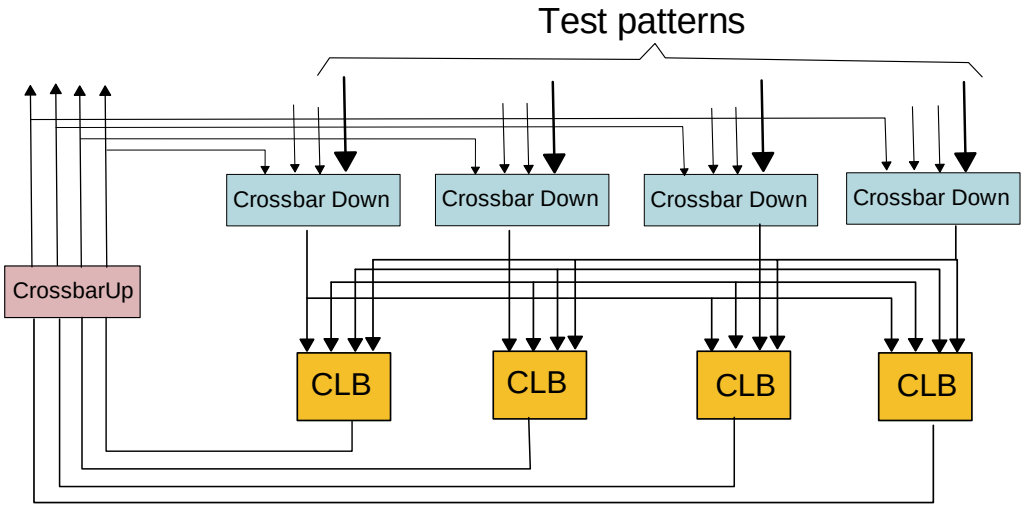
For higher diagnostic resolution, one crossbar 'Down' is tested at a time. Each CLB is then configured to propagate its only active input and the crossbar 'Up' is configured to observe each CLB at the output of the cluster. This process continues until all four crossbars 'Down' are tested one after the other. Although, the crossbar 'Up' in the cluster is tested earlier, it is preferred to keep the crossbar 'Up' configuration unchanged throughout the testing of crossbar 'Down' to avoid any fault masking.



a) First test configuration



b) Second test configuration



c) Third test configuration

Figure 3.10: Test configurations for crossbar 'Down' in a cluster of size 4

The test configuration for CLBs, crossbar 'Up' and crossbars 'Down' is summarized in the form of an algorithm given below.

Algorithm 1: LUT and intra-cluster interconnect test configurations

---

initialize all multiplexers of crossbar 'Down' and crossbar 'UP'

loop 1

select LUT configurations (XOR/XNOR)

apply test patterns and analyze each CLB output

end

loop 2

select next MUX configurations of crossbar 'Up'

apply test patterns and analyze each CLB output

end

loop 3

select one crossbar 'Down': x ((1≤ x ≤4))

loop

select next MUX configurations of crossbar 'Up': x

apply test patterns and analyze each CLB output

end

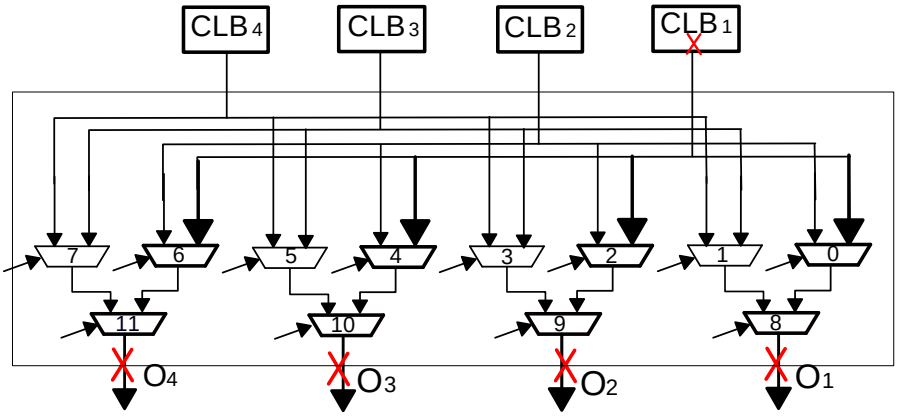
end

The algorithm starts with the initialization of all MUXes in crossbar 'Up' and crossbars 'Down' i.e. each MUX in the crossbar 'Up' selects a unique input, connecting each CLB output to the output of the cluster. Each MUX of a crossbar 'Down' selects only the active input and connects it with the input of the CLB. Then the CLBs are tested in loop 1 where test patterns are applied and test response is analyzed for each test pattern in the ORA cluster. At the end of this loop, the loop 2 starts for testing crossbar 'Up'. The MUXes in the crossbar 'Up' are configured to select their next input. At this stage, CLBs are configured to propagate one of its input. Test patterns are applied and the response of the PUTs is observed. After the completion of crossbar 'Up' testing, the loop 3 starts testing all crossbars 'Down', one by one. During each crossbar 'Down' testing, each input of every MUX is selected at least once.

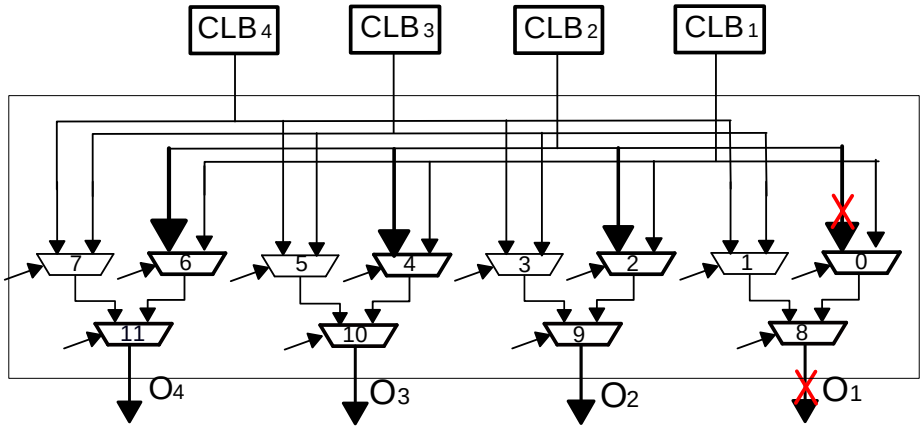
3.6 Optimization of test configurations for crossbar 'Up' and CLB

It is observed that the patterns required for crossbar 'Up' testing can be obtained by configuring XOR/XNOR functions in CLBs. Since CLBs outputs are connected to the crossbar 'Up' multiplexers' inputs, CLBs will be configured to provide the required test patterns to test crossbar 'Up' multiplexers for their every input combination. As we have seen before, the inputs of a MUX must have opposite logic values for a complete test. The inputs of the crossbar 'Up' MUXes coming from CLB, need to be configured to opposite values with respect to the other CLB depending on the PUT in the crossbar 'Up'. We have also seen that programming XOR and XNOR functions in CLB will test completely the CLB. Hence, we can merge the testing of CLBs and crossbar 'Up' in such way that CLBs are tested in the same time as performing crossbar 'Up' configurations. Figure 3.11 shows

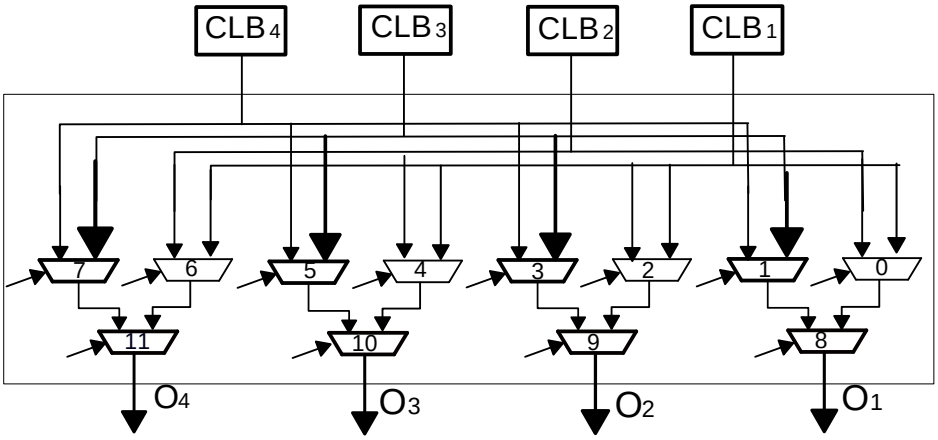
the test configurations required for testing crossbar 'Up' in the case of cluster of size 4. The first configuration tests the first input and the second input is tested in a second configuration and so on. It is important to note that MUXes in crossbar 'Up' are configured such that only one CLB is connected to the cluster outputs (O_1, O_2, O_3, O_4) as shown in Figure 3.11 This is helpful for faults diagnosis which will be explained in later paragraphs.



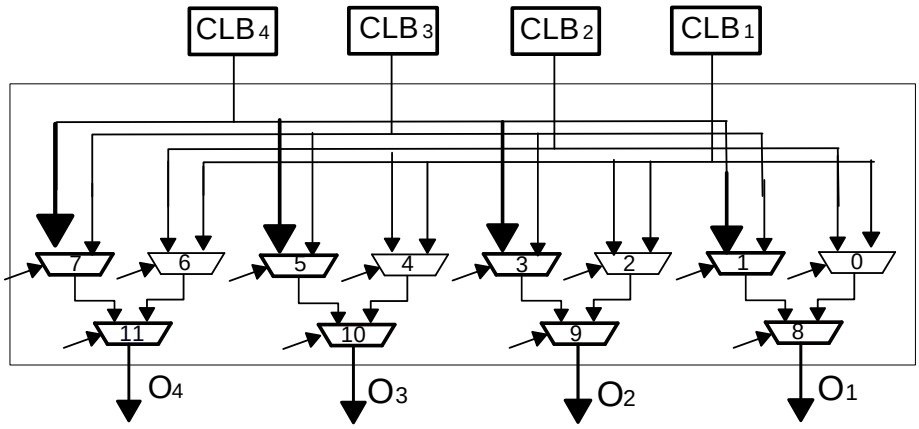
a) First test configuration of Crossbar Up



b) Second test configuration of Crossbar Up



c) Third test configuration of Crossbar Up



d) Fourth test configuration of Crossbar Up

Figure 3.11: Test configurations sequence of Crossbar Up of cluster size 4

TABLE 3.3 TEST CONFIGURATIONS OF LOGIC BLOCK AND CROSSBAR UP FOR CLUSTER SIZE 4

Number of Config.	**Functionality of Logic Blocks**				**CBU /cluster o/p (O_1...O_4)**
	CLB_1	**CLB_2**	**CLB_3**	**CLB_4**	
1	XOR	XNOR	XNOR	Don't care	CLB_1
2	XNOR	XOR	Don't care	XNOR	CLB_2
3	XNOR	Don't care	XOR	XNOR	CLB_3
4	Don't care	XNOR	XNOR	XOR	CLB_4

Table 3.3 presents the functionality of every LUT/CLB in a given cluster during the joint test phase of CLB and crossbar 'Up'. There are some don't care cases, i.e. CLB can

either be configured as XOR or XNOR. The crossbar 'Up' configurations are also shown in the table where a unique CLB is selected in each test configuration for every cluster output i.e. O_1, O_2, O_3 and O_4.

3.7 Generalization of CLB and crossbar 'Up' joint testing

To generalize the CLB and crossbar 'Up' joint testing strategy mentioned above for larger cluster size, we divide the twelve crossbars 'Up' multiplexers in several groups. Consider MUX 0, 2, 4 and 6 in group 'a', MUX 1, 3, 5 and 7 in group 'b' and MUX 8, 9, 10 and 11 in group 'c' (cf. Figure 3.11). Group 'a' and 'b' form level 1 of the 2:1 multiplexer's hierarchy while group 'c' falls in level 2. With given CLB configurations, the configuration of each multiplexer belonging to the same group must be kept identical so that each fault free response of crossbar 'Up' can result an identical value and can be compared at ORA for mismatch.

Joint testing of CLB and crossbar 'Up' is useful only if the diagnostic resolution sustains. And in this context, it is very important to show how this test strategy is able to differentiate whether the fault exists in the CLB or crossbar 'Up'. Each configuration produces identical result of a selected CLB at every crossbar 'Up' output. If a fault exists in a specific CLB, it will propagate at every crossbar 'Up'/cluster output. This scenario is depicted in Figure 3.11(a) where a fault represented by 'x' in CLB_1 is propagated to every output of the cluster. In the ORA, where every output is compared with the output of another BUT, the faulty CLB can easily be identified. Whereas, in case of a fault that exists in crossbar 'Up', only a specific output of crossbar 'Up' will be faulty while other outputs of the crossbar 'Up' will be fault free. It is shown in Figure 3.11(b) where a fault 'x' is observed at only one output of the cluster.

The joint testing of CLB/LUT and crossbar 'Up' allows testing CLBs while testing crossbar 'Up'. Hence, the configurations required for testing CLBs can easily be avoided. Algorithm 1 can be modified for joint test scheme where loop 2 is nested into loop 1, keeping the remaining of the algorithm unchanged. Thus for each cluster, the number of configurations required for testing its CLBs are saved.

Algorithm 2: CLB and intra-cluster interconnect join test configurations

---

initialize all multiplexers of CROSSBAR-DOWN and CBU

loop 1

select LUT configurations (XOR/XNOR)

apply test patterns and analyze each CLB output

loop

select next MUX configurations of crossbar 'Up'

apply test patterns and analyze each CLB output

end

end

loop 2

select one crossbar 'Down' : x ($(1 \leq x \leq 4)$)

loop

select next MUX configurations of crossbar 'Up' : x

apply test patterns and analyze each CLB output

end

end

3.8 BIST structure for cluster

The BIST schemes presented above for testing the CLBs, crossbar 'Up' and crossbar 'Down' in the cluster is implemented using the BIST structure shown in Figure 3.12 where 4x4 mesh FPGA is considered for both test sessions. While developing this BIST structure, we focus on the following features of the BIST

1) Complete test and diagnosis of the cluster in no more than two test sessions.

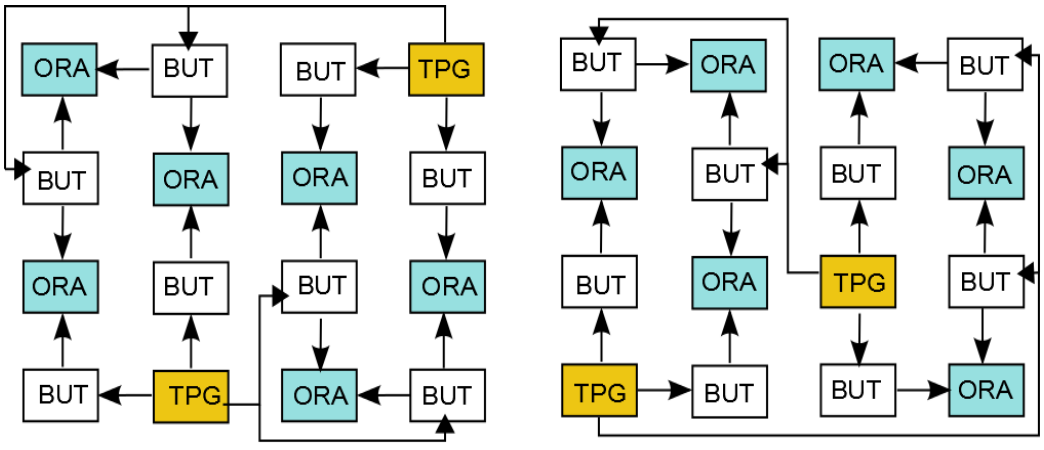
2) Multiple TPGs and ORAs to be used to avoid fault masking.

To achieve such a BIST structure, an efficient BIST strategy is required where the number of clusters configured as TPG and ORA is not more than the number of clusters configured as BUTs in a test session. This can only be possible if the routability of the FPGA allows to implement such structure. In other words, BIST strategy should be

efficient enough to exploit the FPGA regularity and routability well giving less number of test sessions and hence the test configurations.

In this BIST structure, two cluster are configured as TPGs to provide identical test patterns. In session 1, TPG are placed on first and last row while in session 2, on first and second row. In this session, placement of TPGs is such that they can be connected all BUTs in their respective sessions. This session allow to have eight BUTs each. It is interesting to note that six BUTs are analyzed twice in two different ORAs while two BUTs are analyzed only once which forms the limitation of the proposed BIST structure for the cluster. As mentioned earlier, multiple ORAs are used to avoid any fault masking issue if it occurs.

In test session 2, clusters swap their roles i.e. TPGs and ORAs of the session 1 become BUTs of session 2 and vice versa. Similar to the session 1, two TPGs, six ORAs and eight BUTs are configured. Six BUTs are analyzed twice in multiple ORAs and two BUTs are tested only once. In Figure 3.12, connections between TPG, ORA and BUT clusters are shown in 4x4 mesh FPGA and switch box connections are not shown to keep simplicity of the figure. Otherwise, switch box in this 4x4 mesh are also configured to achieve such BIST structure.



TPG: Test Pattern Generator ORA: Output Response Analyzer BUT: Block Under Test

Figure 3.12: BIST structure in 4x4 FPGA mesh

3.9 Switch box test methodology

This section introduces the test methodology of the switch box adapted for mesh FPGA.

To detect any possible stuck-at and bridging fault along a wire, logic '0' and '1' are successively applied at one end of the wire and observed at the other end of it. Typically a number of paths/wires in Upward Mini Switch Boxes (UMSBs) and Downward Mini Switch Boxes (DMSBs) can be selected simultaneously to perform parallel test. For *n* number of selected paths, *n*-bit test pattern is produced by the TPG. These test patterns are then propagated through the selected paths and observed in the ORA for the fault detection. Our test methodology aims to test all the switch box paths exhaustively. For that purpose, we divide the switch box interconnect/paths in the following three groups.

- Group 1: contains all paths from a cluster to another cluster to test UMSBs and a part of DMSBs at 'Level 1'.
- Group 2: contains all paths (interconnects) from the switch box outputs to its adjacent Switch Boxes and paths between DMSBs at both levels, to test a part of DMSBs at 'Level 2' and the remaining part of DMSBs at 'Level 1'.
- Group 3: contains all paths from adjacent switch boxes to the inputs of a switch box, to test the remaining part of DMSBs at 'Level 2'.

Each group is tested in a separate phase where each phase consists of a number of configurations required to test all the paths in that group as mentioned above. The test phases and the group of paths in the switch box is shown in the Figure 3.13.

To perform these test phases, the selection of Paths Under Test (PUTs) in the switch box follows the conditions given below:

Condition 1: All paths selected in a test configuration are equally controllable and observable.

Condition 2: More than 1 fan-out of a net will be considered as disjoint paths and will be selected in separate test configurations. This is due to the fact that a fault on any of the branch/fan-out may affect all other branches and thus can produce masking phenomena.

Condition 3: No assumptions of fault-free interconnect are made. This is to ensure an exhaustive testing without degrading diagnostic resolution and fault masking.

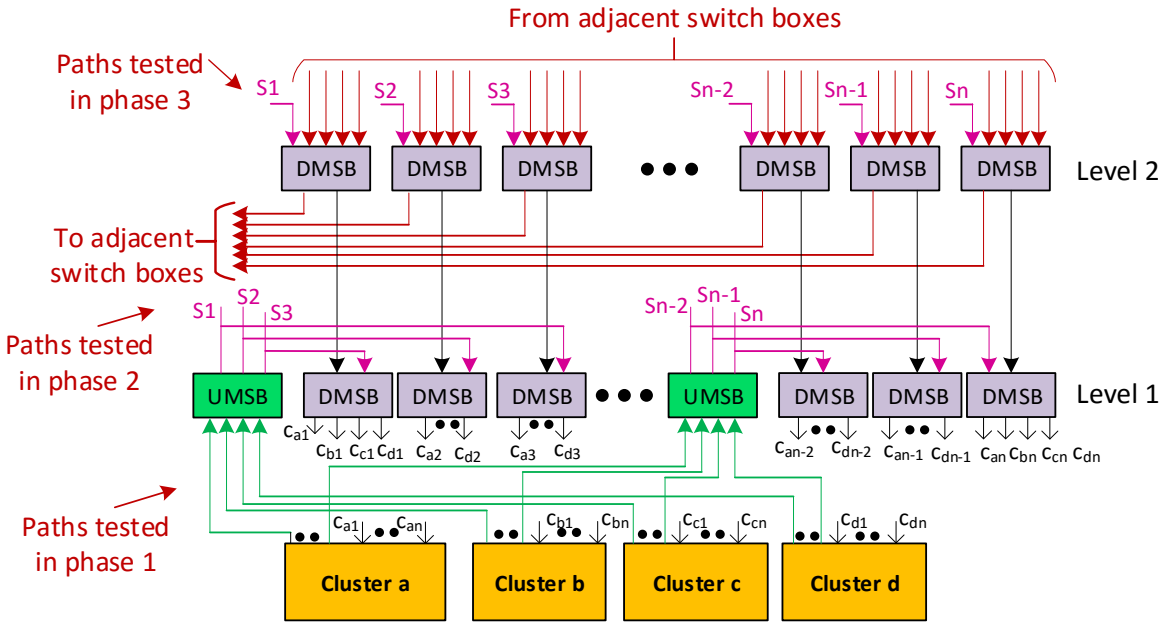


Figure 3.13: Structure of a switch both in mesh FPGA

3.9.1 Test configurations in phase 1

In this phase, the interconnect between the switch box and its 4 adjacent clusters are tested. The number of PUTs selected in each test configuration mainly depends on the number of adjacent clusters and their respective outputs. This is due to the fact that each UMSB has at least one input coming from each adjacent cluster. Therefore, the number of UMSBs at 'Level 1' of the switch box and the number of inputs of each UMSB is given by:

$$No.of\ UMSBs = \frac{No.\ of\ cluster\ outputs}{4} \qquad \text{(eq.3.1)}$$

$$No.of\ UMSB\ inputs = No.of\ adj.clusters \qquad \text{(eq.3.2)}$$

Following conditions 1 and 2 given above for path selection, in this phase we chose PUTs that start from UMSB inputs, passing through DMSB at 'Level 1' and ends at the adjacent cluster inputs. Following condition 2, signals s_1, s_2... s_n between 'Level 1 and 2' are not considered in this phase but will be tested in the next one. The number of PUTs selected per UMSB in each test configuration can be found by the following equation.

$$No.of\ PUTs\ per\ UMSB = \frac{No.of\ adj.clusters\ .\ {No.of\ cluster\ outputs}/{4}}{No.of\ UMSBs} \quad \text{(eq.3.3)}$$

Each UMSB contains N 4:1 multiplexers (MUXes), depending on the number of cluster outputs and the number of UMSBs. For an exhaustive testing of a UMSB, every input of each multiplexer is selected at least once during the test phase. To detect stuck-at faults at the selected input of a multiplexer, test patterns are produced such that the opposite logic values are applied at the unselected inputs with respect to the selected one. Therefore, the test patterns applied at the input of each (4:1) MUX of UMSB must include 0111, 0100, 0010, 0001 for stuck-at 1 and 1000, 1011, 1101 and 1110 for stuck-at 0 faults.

Considering the architecture of the switch box, the above mentioned test patterns for each MUX of the UMSB can only be produced if TPG is implemented in every adjacent cluster. In this way, each MUX of the UMSB receives one bit of test vector from each cluster. Similarly, every adjacent cluster is configured to perform ORA where PUTs are analyzed after each test vector has been applied. A simplified BIST structure for a UMSB testing is shown in Figure 3.14 where only the interconnect considered in this phase is shown. In this BIST structure, some CLBs in each cluster are configured as TPG while some CLBs are configured as ORAs, especially in large clusters which provide sufficient number of CLBs to implement TPGs and ORAs in the same cluster. Following the equation 3.3, the number of PUTs per cluster that can be selected in each configuration becomes 12 in the case of a cluster of size 12. In comparison-based ORAs and in the case of 12 PUTs being tested, 6 CLBs are required (in fact, two PUTs are compared per CLB). Here, we assume that the clusters have already been tested and found fault free in the previous stages using the technique we proposed in section 3.3-3.5 of this chapter.

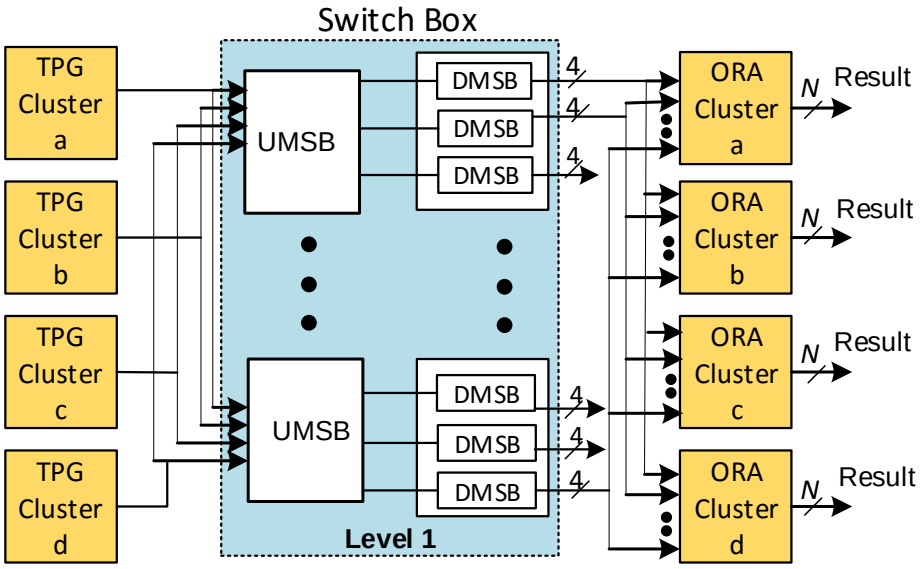


Figure 3.14: BIST structure in phase 1 showing paths under test (PUTs) for UMSBs testing

To perform the test of selected PUTs, DMSBs at 'Level 1' need to be configured such that for each DMSB the only active input comes from UMSB (see Figure 3.14). This configuration of the DMSB at 'Level 1' remains unchanged throughout the testing of UMSBs. Since for testing all PUTs, every MUX of each UMSB is involved, all MUXes in an UMSB as well as all UMSBs can be tested simultaneously. Considering each MUX in an UMSB as a 4:1, we therefore require 4 configurations for its complete testing (one input per configuration). In all 4 configurations, the UMSBs are identically configured and each UMSB test requires 6-bit configuration. Summary of these configurations and the corresponding test patterns is given in Table 3.4. The comparison-based ORA compares two PUTs for mismatch in the single stuck-at fault assumption and provides testing results as '0' when fault free outputs of all the PUTs come from the UMSB .

For pair-wise bridging faults detection, alternate logic values are applied at the multiplexer inputs. Hence, the test patterns applied at each UMSB are 1010 and 0101. In this case, each MUX of a UMSB selects a different input so that alternate PUTs can have opposite logic values. For example, for the test pattern 1010 at each UMSB (hence at each 4:1 MUX in UMSB), the first MUX is configured to selects its first input i.e. '1', the second MUX is configured to selects its second input i.e. '0' and so on. The bridging faults test is also performed using the BIST structure shown in Figure 3.14. Using this structure all MUXes in every UMSB can be tested in parallel. Therefore, only one additional configuration is required for a complete testing of the bridging faults which is shown as the fifth configuration in Table 3.4 along with the applied test patterns. Contrary

to the previous case of stuck-at faults, PUTs having alternate logic values result in logic '1' as a fault free response for the comparison and logic '0' in the case of bridging fault detection.

TABLE 3.4 CONFIGURATIONS AND TEST PATTERNS FOR A UMSB AT 'LEVEL 1'

No. of Configurations	UMSB Config. bits	Test patterns at UMSB	Fault free output	Faulty output
1	000000	0111 1000	0	1
2	010101	0100 1011	0	1
3	101010	0010 1101	0	1
4	111111	0001 1110	0	1
5	000110	0101 1010	1	0

3.9.2 Fault detection and diagnosis in phase 1

Considering an example of a cluster size 12, we have 3 UMSBs at 'Level 1', each of it having three 4:1 MUXes. As discussed earlier, all 3 UMSBs are tested simultaneously. For stuck-at faults detection, the first configuration i.e. 000000 (cf. Table I) of a UMSB allows the three MUXes to select their first input. Applying '0111' test pattern in this configuration, all 4 PUTs in the UMSB gets at the end of path identical logic values '0'. Similarly, if all three UMSBs are configured identically, the resulting 12 PUTs will propagate a logic value '0' for '0111' test pattern and '1' for '1000' test pattern. These 12 PUTs are analyzed in the ORA cluster, where each CLB compares 2 PUTs for any mismatch e.g. PUT_1 and PUT_2 are compared in CLB_1, PUT_3 and PUT_4 are compared in CLB_2, etc. The comparison result is stored in the CLB flip-flop and extracted at the end of each test sequence using already available scan-chain. Once a faulty PUT is detected, it is imperative to locate the fault and understand whether it lies in the UMSB or DMSB of 'Level 1' since every PUT involves both UMSB and DMSB. It was mentioned earlier that in this phase, the DMSB configuration remains unchanged during the UMSB testing. It implies that if a fault persists in a specific output of the analyzer during the test phase, that means that the fault exists in DSMB, otherwise it is in UMSB. The granularity of the

diagnosis can be brought to the specific input/output at the MUX level in DMSB/UMSB by cross examining the current test configuration and the ORA output sequence.

Similarly, the second configuration in this phase selects the next input of all four multiplexers of UMSBs. Dedicated test patterns are given in Table 3.4. They are applied and the corresponding result is observed. This process continues until the completion of stuck-at fault detection in 4 configurations.

For bridging fault detection, opposite logic values are applied to the adjacent inputs of the UMSB. In this configuration, all three MUXes in a UMSB select a different input i.e. MUX_1 selects its first input; MUX_2 selects its second input and so on. In this case, logical mismatch among PUTs in an ORA indicates the fault-free response and vice versa.

3.9.3 Test configurations in phase 2

In this phase, interconnects between switch box outputs and its adjacent switch boxes, and interconnect between DMSBs at 'Level 1' and 'Level 2' are tested. To simplify the testing strategy, test configurations are divided into different test sessions. In every test session, a unique BIST structure is formed to test certain PUTs. A complete testing procedure for such PUTs may require more than one test session due to the architectural constraints.

In this phase, the number of PUTs per configuration depends on the channel width of the FPGA architecture.

$$No.of\ DMSBs\ at\ 'Level2' = \frac{FPGA\ Channel\ Width}{2} \qquad \text{(eq.3.4)}$$

$$No.\ of\ PUTs/adj.SB = \frac{No.\ of\ DMSBs\ at\ 'Level\ 2'}{No.\ of\ adj.\ switch\ boxes} \qquad \text{(eq.3.5)}$$

Equation 3.5 gives the number of DMSBs at 'Level 2' such that there is a dedicated DMSB for each unidirectional wire in the routing channel coming from the adjacent switch boxes. The number of PUTs per adjacent switch box that can be selected in each test configuration is given by the equation 3.5. In the following, BIST structures are developed in which some of the adjacent clusters are configured as TPG while some as ORA as compared to phase 1 where TPG and ORA are implemented in the same clusters.

We form two sets of PUTs: 1) PUTs starting from DMSBs at 'Level 2' (i.e. inputs s1, s2....), passing through DMSBs at 'Level 1' and ends at the ORA cluster. 2) PUTs starting from DMSBs at 'Level 2' (i.e. inputs s1, s2...), passing through one of the adjacent switch

box and ends at the same ORA cluster. Figure 3.15 shows the BIST structures for these test sessions where connections between adjacent clusters (a, b, c and d) and adjacent switch boxes (SBa, SBb, SBc and SBd) involved in the BIST structures are shown in full/bold whereas PUTs are shown in red.

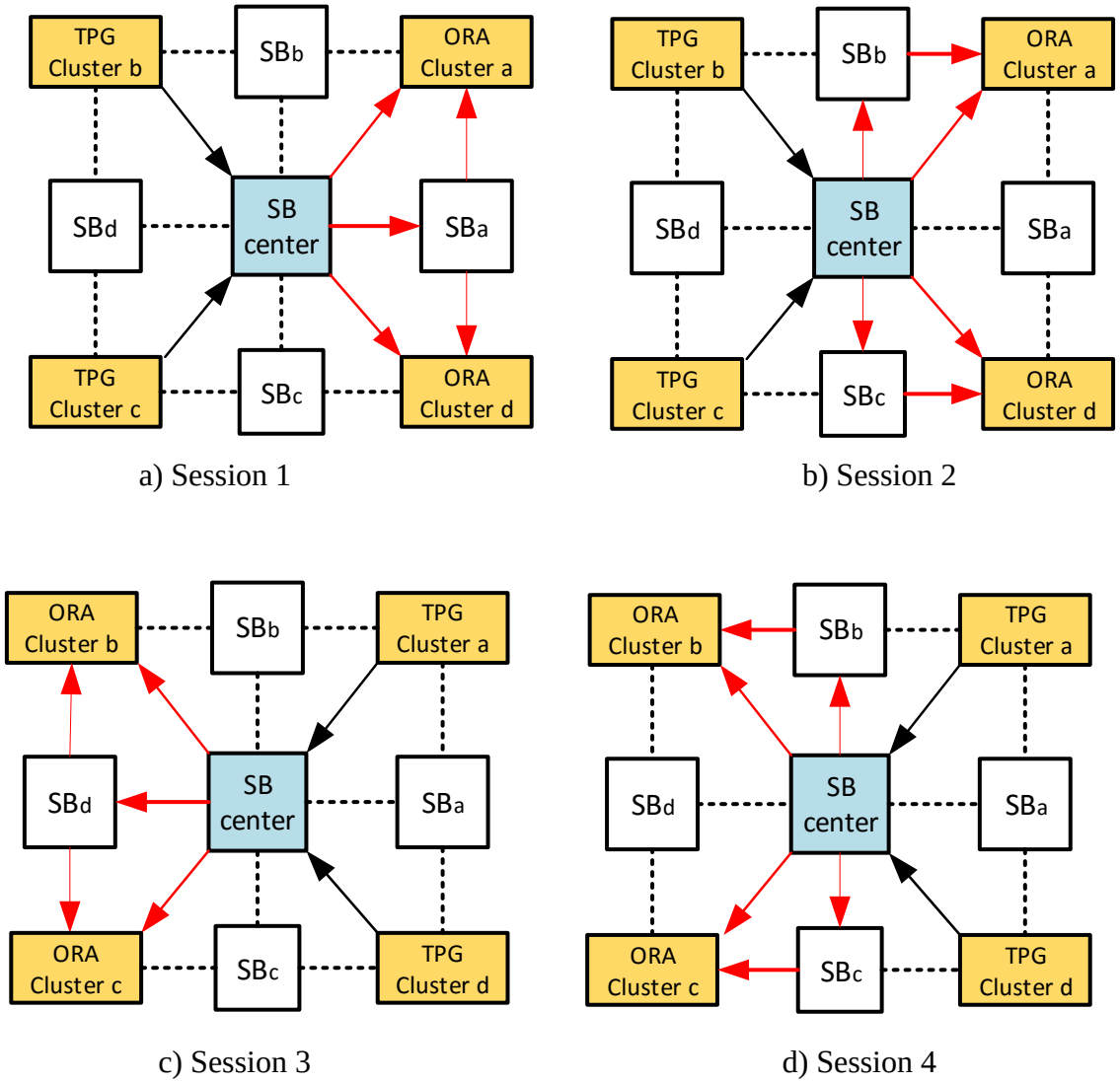


Figure 3.15 : BIST structures in phase 2 showing paths under test (in red) for testing Switch Box (SB) outputs

In the first test session, two among four adjacent clusters are configured as TPG to produce identical test patterns. UMSB at central switch box (SB_{center}) is configured such that these test patterns are supplied at s_1, s_2...s_n inputs of the DMSBs at 'Level 2'. From here, these test patterns are applied to both sets of PUTs. The DMSBs at 'Level 2' are configured to select inputs s1, s2...(where test patterns are applied), for all its corresponding outputs.

Thus, for the PUTs of set 1, each DMSB at 'Level 1' selects the only input coming from DMSB at 'Level 2' and make it observable in the ORA e.g. 'cluster a' as shown in Figure 3.15(a).

For the PUTs of set 2, output signals of SB_{center} are passed through the adjacent SBa and observed at the same ORA 'cluster a'. Similarly, for ORA 'cluster d', the PUTs of set 2 remain the same (through adjacent SBa) but PUTs of set 1 are different.

3.9.4 Fault detection and diagnosis in phase 2

In this phase, the ORA clusters compare the following PUTs:

1) PUTs coming from SB_{center} (Let us call them as 'set 1 PUTs') and
2) PUTs coming from adjacent switch boxes (Let us call them as 'set 2 PUTs').

Since both sets of the PUTs are sensitized by identical test patterns, any mismatch observed at the end of the path can detect the fault. However, detecting the fault does not necessarily mean that it is easy to locate it by just observing the output sequence of the ORA cluster in one session. In order to locate the fault, one set of PUTs need to be changed keeping the other set as it is as shown in Figure 3.15(b) session 2. In both sessions, the PUTs of set 1 (located between SB_{center} and ORA cluster) remain unchanged which helps to determine the fault location. If the fault persists at the same output of an ORA cluster, it indicates the existence of fault in the specific PUT of set 1; otherwise the fault exists at the output between SB_{center} and adjacent switch box (PUT of set 2). A prominent advantage of the proposed method is that it does not require the assumption of fault free adjacent SB which fulfills the condition 3 for PUTs selection. But it does require two test/diagnosis sessions to locate the fault. At the end of test session 1 and 2, TPG and ORA clusters swap their roles in order to complete the testing of the output interconnect between SB_{center} and the remaining two adjacent clusters as shown in Figures 3.15 (c) (d) sessions 3 and 4.

3.9.5 Test configurations in phase 3

In this phase, the input interconnect of SB_{center} coming from adjacent switch boxes are tested. A BIST structure is developed (cf. Figure 3.16) following the same logic used for testing the output interconnect in phase 2. In this phase, a PUTs starts from the input of SB_{center}, passes through DMSB at 'Level 2 and 1' and end at the ORA cluster. Two TPGs are implemented in two adjacent clusters and the test patterns are applied to SB_{center} through one of the adjacent switch box. Since a single TPG cluster can supply only a limited number of test patterns to the switch box, therefore, two TPG clusters are used to

produce the required test patterns. Each DMSB at 'Level 2' is configured to select the only input coming from this adjacent switch box. Similarly, DMSBs at 'Level 1' are configured to select the input coming from 'Level 2'. Finally, the PUTs are compared in the CLBs of the ORA clusters. Finally four sessions are required to complete the testing of input interconnect of SB_{center}, as each session involves only one adjacent SB. The number of PUTs in this phase can be calculated by using equation 3.5.

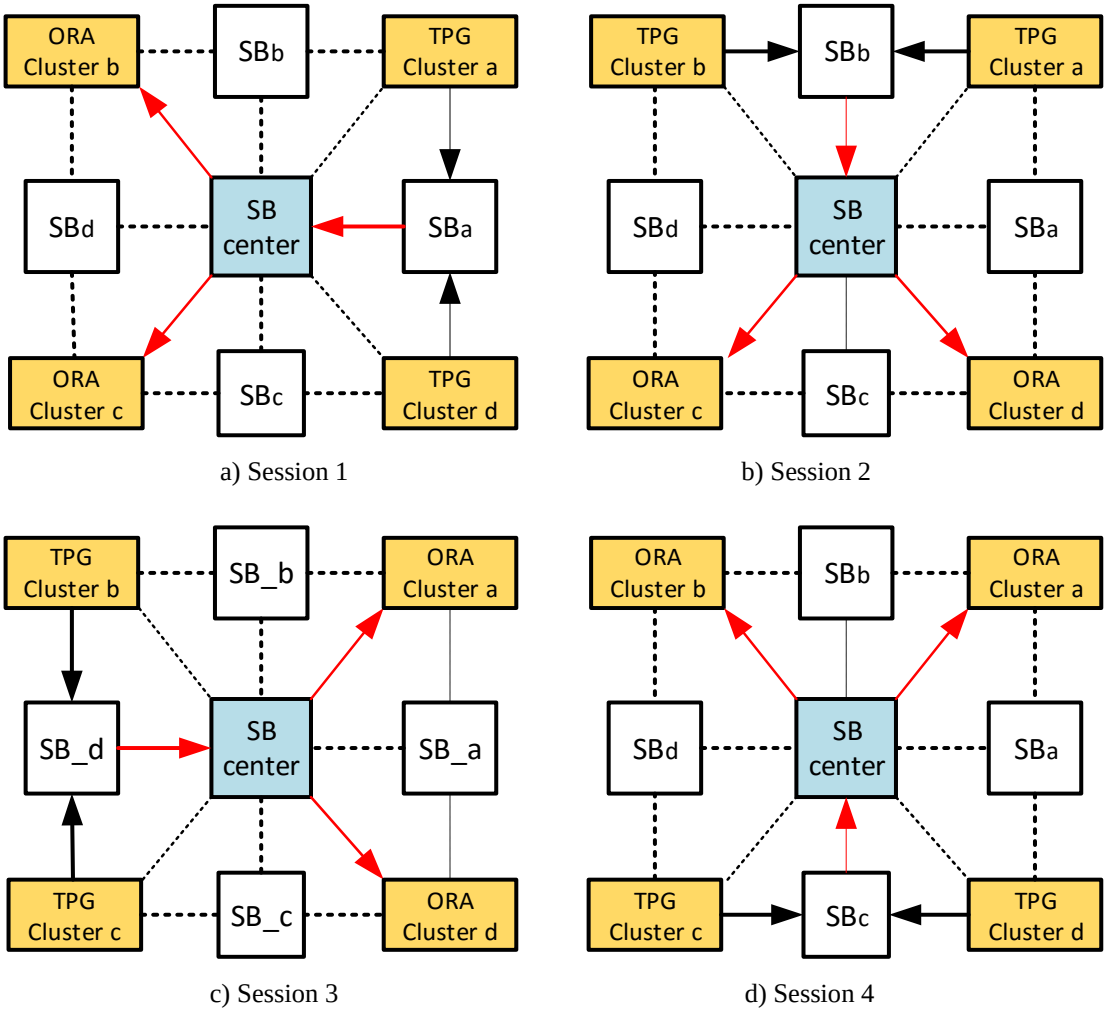


Figure 3.16: BIST structures in phase 3 showing paths under test (in red) for testing Switch Box (SB) inputs

3.9.6 Fault detection and diagnosis in phase 3

In this phase, each ORA cluster analyzes a certain number of PUTs but these PUTs are checked only once as compared to previous phase. The reason behind is that every PUT is unique and the DMSB at 'Level 1 and 2' have already been tested in previous phase and considered fault-free. Therefore, multiple comparisons of a PUT are not required for fault diagnosis. When a fault is detected, its location is determined at the

switch box inputs by manipulating the mismatch at the ORA output sequence and the current test configuration.

The prominent feature of BIST structures developed here is that Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) are implemented in clusters adjacent to switch box, making a unit of 2x2 BIST structure. In this way, this unitary structure can be implemented on any FPGA array size, by performing parallel testing of N (2x2) arrays. The BIST structure itself defines the number of test configurations required for complete testing. Therefore, an efficient BIST structure allows having the maximum number of PUTs in a test configuration and gives the maximum fault coverage and high diagnostic resolution.

The efficiency of the BIST schemes is evaluated by the number of test configurations required to achieve the targeted fault coverage. These test configurations then determine the test time, hence the test cost as in the case of FPGA BIST. We find it important to present the results (chapter 6) after explaining the CAD flows for the implementation and automation of the proposed schemes in the next chapter.

3.10 Conclusion

In this chapter, we discussed the test methodology for CLBs, intra-cluster crossbars and the switch box in a mesh FPGA. Each block of the FPGA requires a dedicated BIST structure and related test configurations. Our test strategy first tests the CLBs, then intra-cluster interconnect and then switch box. In the proposed BIST schemes, we assumed that SRAM used for configuration cells have already been tested and found fault free. For testing the cluster, the BIST strategy presented above requires only two test sessions which is the least minimum sessions in FPGA BIST. It shows that the BIST strategy utilizes the FPGA architecture well and pushes the routability of the architecture to the point where it can implement BIST structure by utilizing only adjacent blocks. Similar is the case with switch box testing where 2x2 unitary BIST structure is developed to completely test and diagnose a switch box. Moreover, the sparsely populated crossbars prove high routability of the FPGA architecture considered here. The diagnostic resolution is aimed at MUX level during the testing procedures of CLBs and interconnects. An optimized scheme for joint testing of CLB and crossbar up is also presented which helps testing CLB and crossbar up in parallel without loss of diagnostic resolution. The resulting fault coverage and the number of configuration of each case will be discussed in chapter 6 after the implementation using CAD tools.

Chapter 4

Test and diagnosis schemes for defect tolerant FPGAs

4.1 Defect tolerance in FPGAs

With respect to the defect tolerance, FPGAs have attained a central focus due to their reconfigurability which enables to bypass the defective areas and implement the application on defect-free resources. There are several techniques for repairing FPGAs in case of permanent faults detection. Most of these hardening schemes resort to redundancy and can be classified into software-based and hardware-based techniques.

4.1.1 Software-based hardening

Software-based hardening requires no modification in the basic FPGA architecture. Defect tolerance is provided solely through place and route mechanism around defects. When a fault is detected and located, the application is reconfigured on the spare resources in the FPGA. This kind of software-based hardening generates additional reconfiguration time and memory and induces considerable area overhead in terms of spare resources in FPGA. In some cases e.g. [Huang 2006], this application shifting mechanism has been made time-efficient by providing pre-compiled alternate place and route solutions. In addition to this, memory cost has been reduced by producing differential configuration in which the application is shifted to the spare neighboring column of the FPGA array.

4.1.2 Hardware-based hardening

Hardware-based hardening involves modification of original FPGA architecture to make it defect-tolerant. In [Doumar 1999], defect tolerance is achieved by automatic shifting of configuration data bits from defective to defect-free FPGA resources. It requires neither configuration data to be loaded from off-chip memory nor any intervention from the user. A small modification is made in the FPGA architecture where SRAM cells are altered such that they can shift the stored data to the neighboring cells.

This approach saves the reconfiguration time but costs the spare resources and alteration of SRAM cell contents.

In some cases, hardware redundancy is achieved by adding spare logic/interconnection resources that are swapped with the defective ones. The swap time is typically less than the time needed to generate a new placement and routing solution. This hardware-based redundancy can be implemented at different granularity levels.

Coarse-grain redundancy [Yu 2005] uses the redundancy of rows/columns in mesh FPGA architecture. Defects are avoided by substituting the supplementary row/column for the defective one. As a matter of fact, the coarse-grain redundancy is suitable for tolerating clustered defects. However, it cannot tolerate multiple unclustered distributed defects.

Fine-grain redundancy [Yu 2005] employs redundant routing resources by adding more switches in the switch box. In order to successfully bypass a defect, a defect map should be either stored on-chip in non-volatile storage, or in an off-chip database indexed using a unique on-chip ID. When the FPGA is being programmed, defect avoidance is performed according to the defect map. A defect is avoided by shifting individual signals. A shift-avoid and shift-restore mechanism is fully described in [Yu 2005]. This mechanism eliminates the need to re-route the whole application and thus reduces the correction time. Unlike the coarse-grain redundancy, fine-grain redundancy technique tolerates multiple randomly distributed defects. In terms of area, coarse-grain redundancy engenders less overhead for low-density defects. Whereas, fine-grain redundancy incurs a fixed area overhead of 50% for all array sizes, thus tolerating an increasing number of defects as the FPGA size grows at a constant cost. Redundancy can be implemented at a finer level. For instance, spare connections can be added inside the switch block to tolerate one transistor defect per switch block [Doumar 2000]. However, this approach costs an average of 3% delay penalty and a partial modification of original data.

Most of the previous work on FPGA defect tolerance was done without taking FPGA's testability into consideration. Usually, the focus remained on improving the robustness of the FPGA by introducing redundancy without working out any trade-offs. Therefore, the test cost of a defect tolerant FPGA has not been well analyzed. This chapter discusses the testability aspects of some efficient defect tolerant techniques applied on the FPGA considered in this thesis. For that purpose, we used three different defect tolerant FPGA architectures developed by our project partners. These defect-tolerant schemes are applied at different blocks of the FPGA and are classified as logic and intra-cluster interconnect level redundancy. One of the hardening techniques is purely

hardware-based in which redundancy is applied at logic level. Other two techniques are applied on the intra-cluster interconnects and involve both hardware and software based hardening. On one hand, they are considered hardware-based as they require fine-grain redundancy while on the other hand, they are software-based because both techniques require remapping and re-routing of the application such that defective interconnect resources can be bypassed. A brief overview of each technique is presented below.

4.2 Redundancy in logic blocks

A classical way of hardening the CLB design is to triplicate the whole LUT-4 and use a Triple Modular Redundancy (TMR) voter downstream [Ban 2010] as shown in Figure 4.1. The advantage of this approach is the equivalence of many defects within the same LUT-4 to a single defect at the output of that LUT. In this case, the Triple Modular Redundancy (TMR) voter adopting a majority voting strategy outputs the correct value. However, triplicating the whole CLB has two major drawbacks. First, if two defects happen at the outputs of two modules, a wrong value will be voted. Second, TMR applied at the CLB level produces too much area overhead. Indeed, there are thousands of CLBs in an FPGA. Thus, in order to get a good trade-off between robustness and area overhead, we used redundancy at a finer granularity level, such as the MUX2.

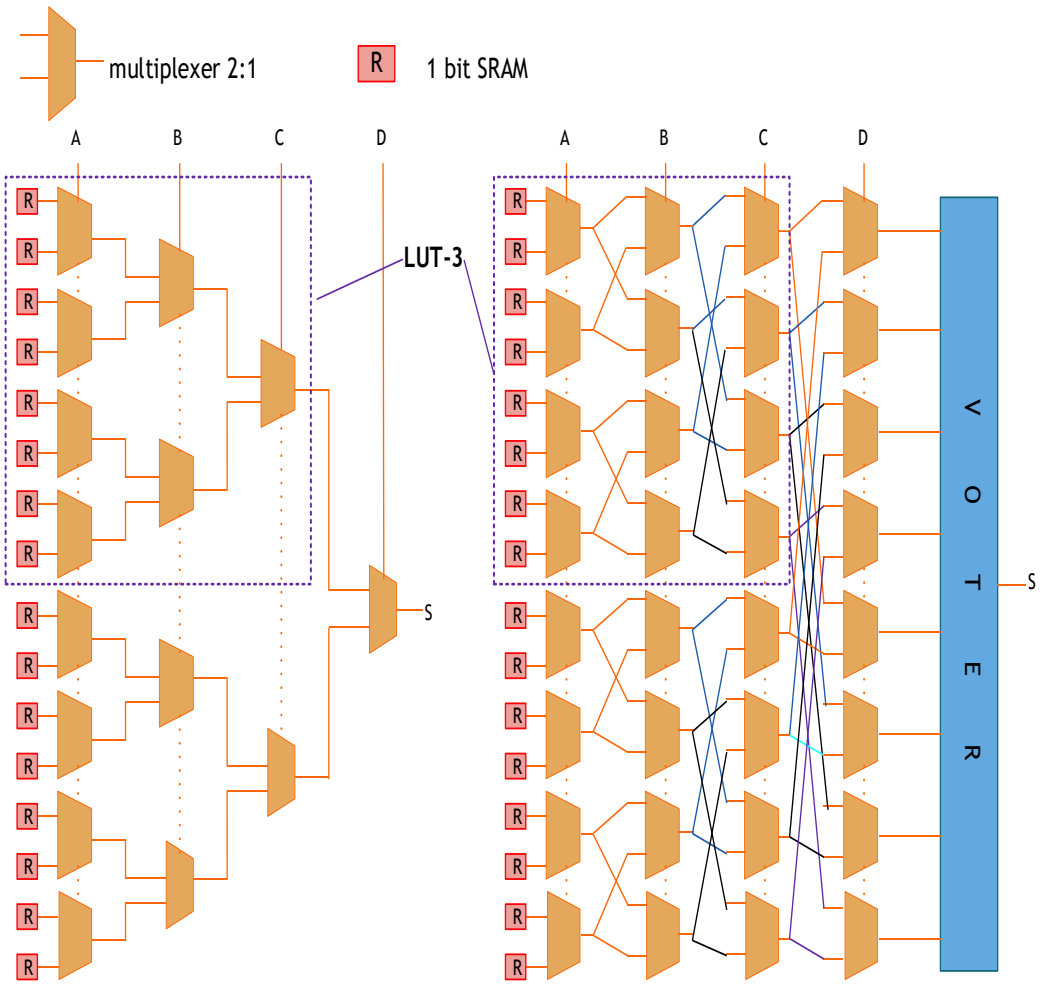


Figure 4.1: a) Simple LUT-4 b) LUT-4 with TMR [Kyria 2009]

Among the large amount of architectures reported in the literature, we have adopted one developed by one of our project partner, also, described in [Dhia 2012]. Such architecture is called *Butterfly* and, according to the authors, has proved to be more robust than the conventional LUT. In *Butterfly* architecture, a LUT-N has N stages, each stage containing 2^{N-1}Mux2s. The LUT-N output is obtained by a bitwise majority voter in the last stage. In the case of the LUT-4, an 8-input voter is needed, which is obviously more costly in terms of area, power and delay than a TMR voter. To simplify the original *Butterfly* structure with the purpose of having a mere TMR voter downstream, a modified version is proposed in [Dhia 2013]. Consequently, some Mux2s had to be removed. Figure 4.2 represents the modified version of the *Butterfly* design. As far as the voter is concerned, a fault-tolerant TMR voter introduced in [Ban 2010; Naviner2011] is used and represented in Figure. 4.3. This voter is tolerant to single faults.

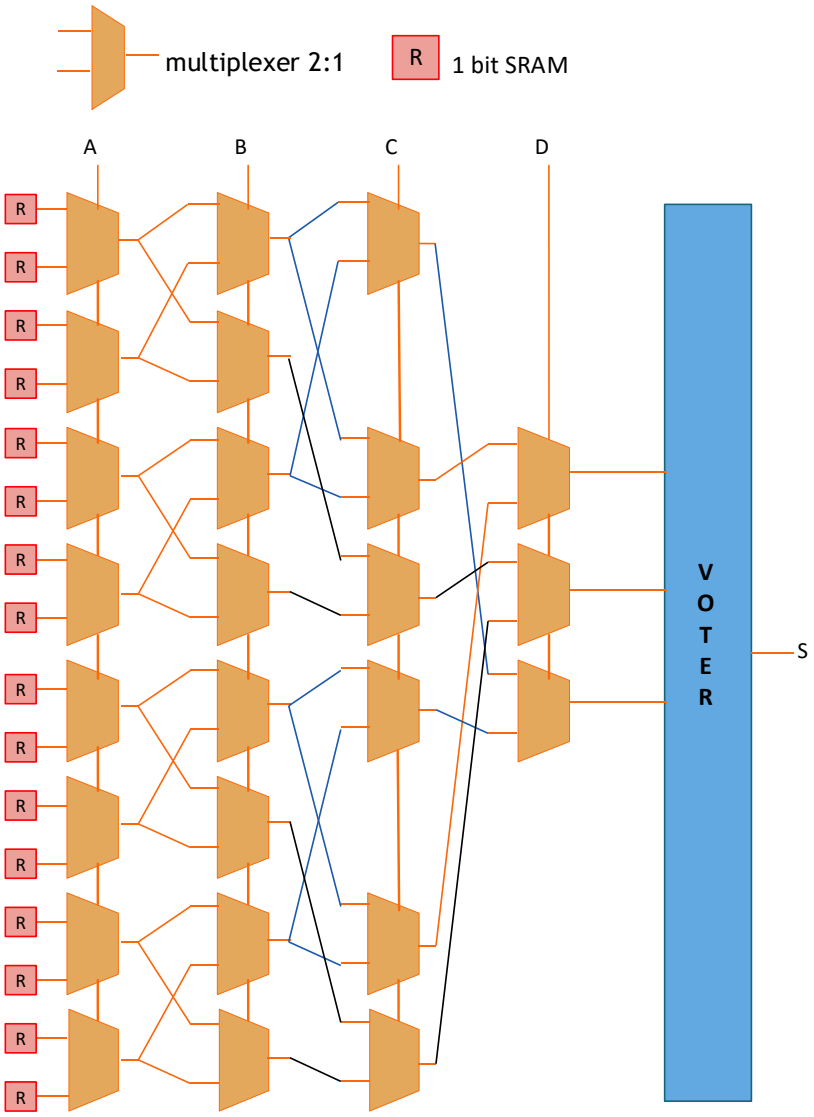


Figure 4.2: LUT-4 modified Butterfly design [Dhia 2013]

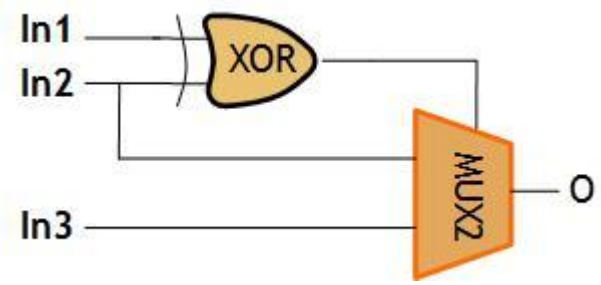


Figure 4.3: Fault tolerant voter [Ban 2010]

4.3 Test configurations for *Butterfly* LUT/CLB

For the *Butterfly* architecture, we notice that the number of inputs and outputs of a *Butterfly* LUT remain identical to that of a typical LUT. Hence, the SRAM cells or the number of configuration bits remain unchanged. For that reason, we use exactly the same BIST strategy developed for simple LUT in chapter 3 since that strategy is already exhaustive. Here we assume again that SRAM cells are fault free. In LUT mode of operation, XOR function is configured and the 4-bit test pattern is applied. In the next configuration, XNOR is configured and test patterns are applied. The test patterns are produced by using 4-LFSR as in the previous case.

The BIST structure developed for simple LUT is also used for *Butterfly* LUT. In this structure, single TPG is used to feed multiple BUTs and a comparator-based ORA is used to analyze the output responses as shown in Figure 4.4.

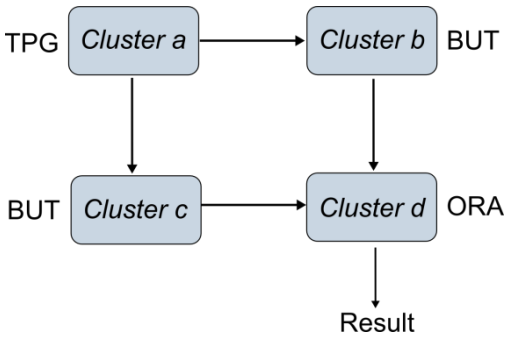


Figure 4.4: BIST structure for CLB/LUT

4.3.1 Diagnosis in *Butterfly* LUT

As mentioned before, in B*utterfly* LUT, some MUXes are removed from the original TMR LUT. This simplification gives rise to some redundant paths in the LUT. As a result faults masking is produced in those paths. Similarly, the majority voter enhances this inherent masking ability and the fault coverage deteriorates even if exhaustive test patterns are applied. An example is shown in Figure 4.5 where LUT is configured as XOR and the test pattern (0111) is applied, thus attempting to propagate an injected fault (SA0) to the output. The node where fault is injected has a single propagation path to the voter. Thus, this SA0 fault is masked and cannot be detected because the majority voter gives identical output i.e. '1' both in case of fault and when the circuit is fault free. (In Figure 4.5 the faulty output is shown in parenthesis such as (0)).

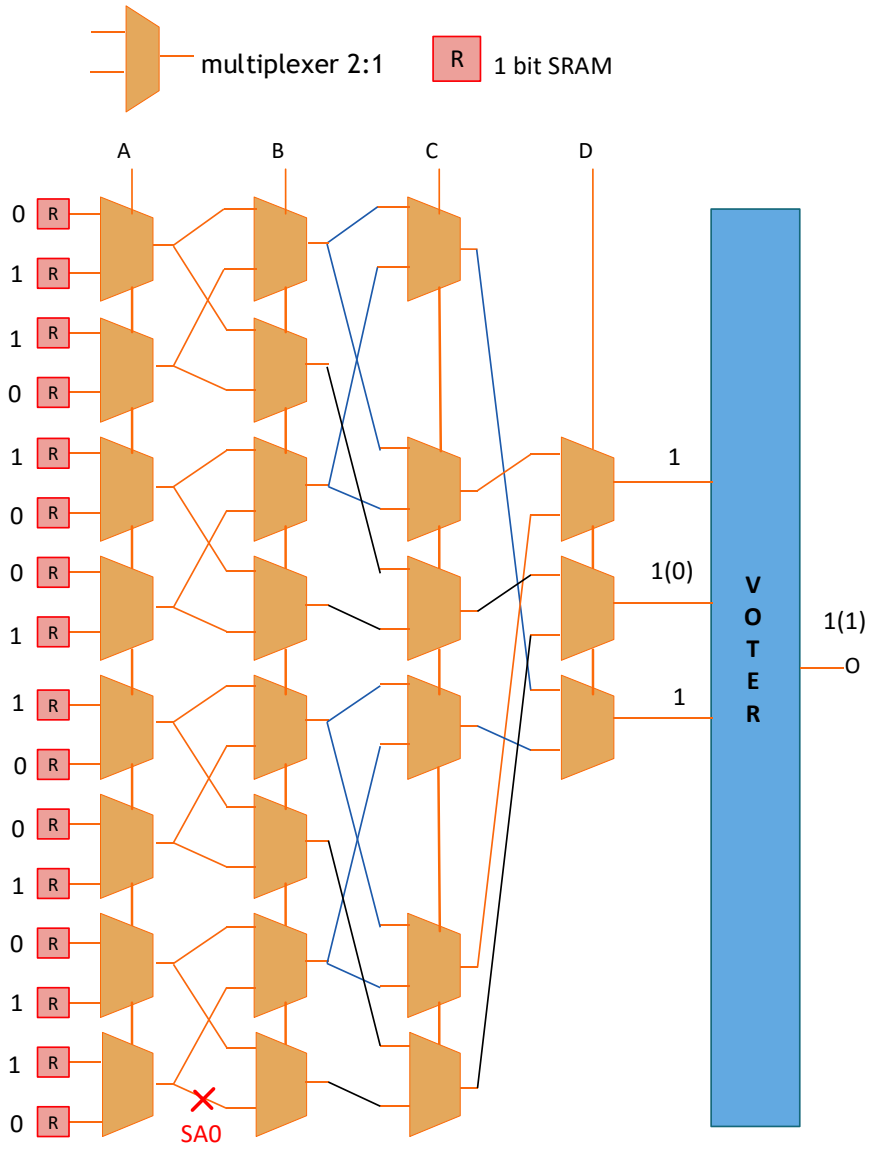


Figure 4.5: Fault masking example in *Butterfly* LUT

4.4 Redundancy in interconnect

There are many ways in which redundancy can be applied at interconnect level. We employ two different techniques for redundancy at intra-cluster interconnect level i.e. crossbar 'Down' and crossbar 'Up':

1) Fine Grain Redundancy (FGR)
2) Distributed Feedback (DF)

These redundancy techniques were integrated in the FPGA architecture considered here by [Amouri 2013]. In the following, brief overview of these redundancy techniques and the test configurations developed for the defect tolerant cluster is given.

4.4.1 Fine Grain Redundancy (FGR)

In the crossbars 'Up' and 'Down', four levels of Mux2s are added: two levels of Mux2s upstream to avoid the defect by shifting the signal, and two other levels downstream to restore the signal. Figure 4.6 depicts a crossbar 'Down' hardened with FGR technique. The crossed Mux2 in the Figure represents a defective Mux2. It was meant to connect the input I5 to the output O8. The FGR upstream allows re-routing the I5 signal to the neighboring Mux2. Then, the FGR downstream allows restoring the signal that will be connected to O8. Nonetheless, the I5 signal can be re-routed only if the neighboring Mux2 is a spare resource.

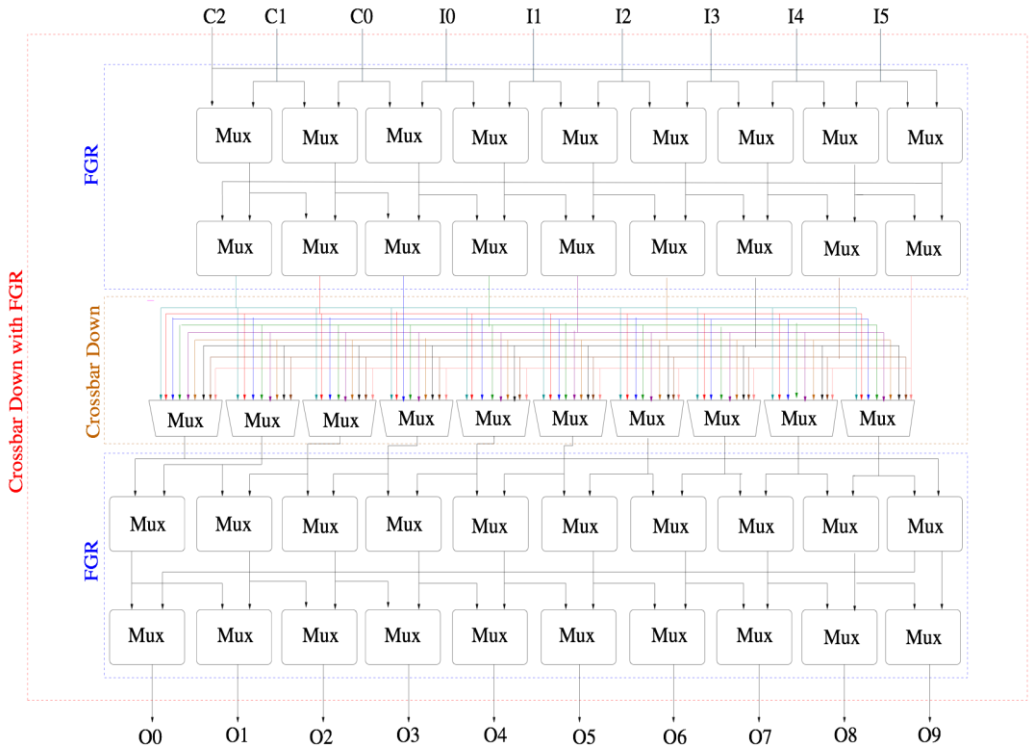


Figure 4.6: Crossbar 'Down' hardened with FGR technique [Amouri 2013]

4.4.1.1 Test configurations for crossbar hardened with FGR

To develop a test strategy, we divide the crossbar hardened with FGR technique in three functional blocks. 1) Basic crossbar block 2) upstream FGR block 3) downstream FGR block. The test strategy is to test the added upstream and downstream blocks separately from the basic block. In this way, we will be able to use the test configurations

already developed for the basic cross bar block. In this regard, three phases are defined. In phase 1, the basic crossbar block is tested. In phase 2 and 3, the up and downstream FGR blocks are tested, respectively.

Test phase 1: In order to test basic block, upstream FGR block is configured such that every input is selected in at least one MUX. In this way, crossbar inputs will be connected to the basic block as in the crossbar without FGR. Similarly, downstream FGR block is configured to connect each output of the basic block to the output of the crossbar. In this configuration, any two MUXes in the down and upstream will select the common input. Having the configuration of up and downstream FGR blocks, the basic block is tested exhaustively for all its configurations as explained in chapter 3. Throughout the basic block testing, up and downstream FGR block's configurations remain unchanged.

Test phase 2: Once the basic crossbar block is tested completely, upstream block is tested. During this phase, the basic block is selected to have any of its fault free configurations (obtained from previous phase). The downstream block keeps its configuration as in the previous phase as well. There are two levels of 2:1 MUXes in upstream block requiring 4 configurations for a complete testing. It relates to the example given in chapter 3 where crossbar up of two hierarchical levels of MUXes is tested with 4 configurations.

Test phase 3: In this phase, downstream FGR block is tested. Similar to upstream block, downstream block also consists of two levels of 2:1 MUXes. Thus, the number of configurations required to test this block is also 4. During the testing of downstream block, the basic block's configuration remains unchanged. However, upstream block is configured back to the same configurations of phase 1 which means that every input is selected in at least one MUX of the upstream block. It is critical to apply test patterns which are generated in a proper sequence for testing a two hierarchical level of MUXes in the downstream block.

4.4.1.2 Diagnosis of crossbar hardened with FGR

If a fault is detected during any phase, it is assumed that the fault exists in the block which is currently under test. It is critical to have this assumption because a block cannot be both controlled and observed only using its inputs and outputs. This is also most probable assumption because the other two blocks keep their configurations unchanged throughout the testing phase of a block. Thus, if the fault is detected in one test configuration and the other configurations in that test phase are fault free, it indicates that the fault could be in the block currently under test.

During the test of the other blocks, if the same path is detected faulty in one of its configuration, then the location of the fault can easily be confirmed just by observing the outputs sequence and the crossbar configuration. Hence, a correct diagnosis is made at the end of all test phases.

4.4.2 Distributed Feedback (DF)

It has been discussed in chapter 3 that each crossbar 'Down' in the cluster has some inputs coming from crossbar 'Up'. These inputs are the local feedback signals from CLBs which are uniformly distributed among the sparsely populated crossbar 'Down' within the cluster. Consider a situation in which a feedback signal happens to be routed by a defective Mux2 in the crossbar ‘Down’. In this case, the connection will not be possible and the cluster may become unusable. To avoid this kind of problem, [Amouri 2013] proposed a solution in which one (or more) feedback signal are distributed to all crossbars ’Down’. Thus, a defective connection can be rerouted through another crossbar ’Down’. We call them Distributed Feedbacks (DF).

An example of a cluster enriched with DFs is shown in Figure 4.7. Among the cluster’s twelve outputs, eight are feedback in pairs to four crossbars ‘Down’, and the four other outputs (drawn in red) are feedback to all crossbars ‘Down’.

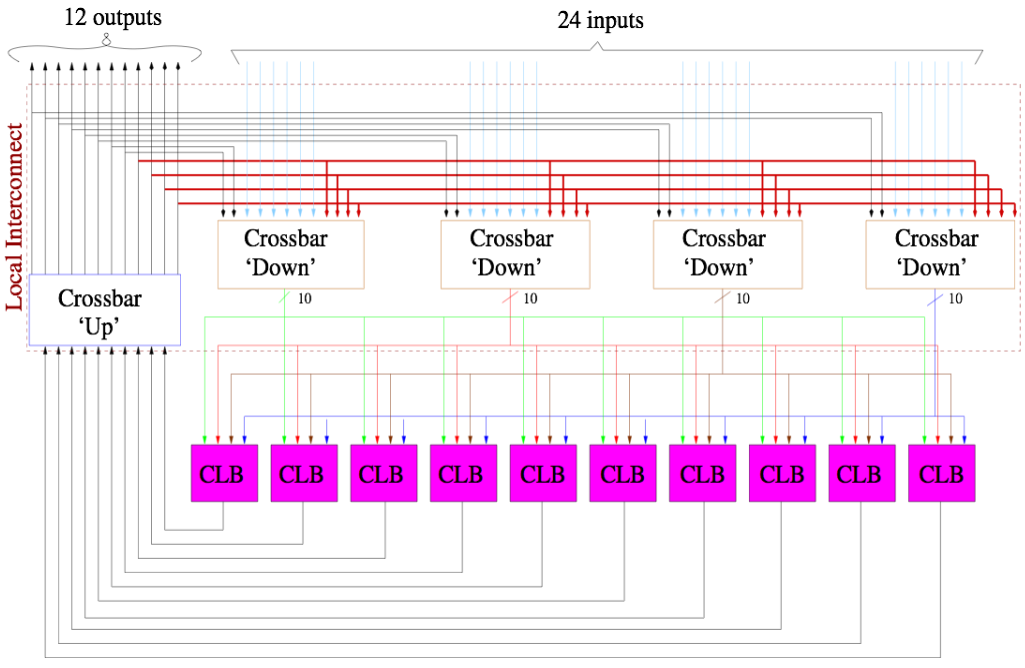


Figure 4.7: Crossbar 'Down' hardened with FGR technique [Arwa 2013]

In the DF technique, the possibilities to route the same signal are multiplied by 4. As a result, the number of inputs per crossbar ‘Down’ increases, as well as the size of all

multiplexers. Like in the aforementioned FGR approach, a defect map is required to bypass the defects. While configuring the FPGA, the defective Mux2s inside the crossbar ‘Down’ are bypassed and all the feedbacks are routed to the other defect-free Mux2s. The DF method is focused to improve the connectivity between CLBs within a cluster. The impact of introducing DF technique on the cluster area, routability and test cost will be discussed in chapter 6.

4.4.2.1 Test configurations for Distributed Feedbacks

The strategy developed for testing the crossbar 'Down', crossbar 'Up' and CLBs of a basic cluster in chapter 3 can be used to test the cluster enriched with DFs. Some extra configurations are required to test the added crossbar 'Down' inputs as DFs. For testing the feedback paths, crossbar 'Down' is reconfigured to select 'feedback' input after the configuration in which 'regular' input is selected for test pattern. In this way, the cluster output in one configuration becomes the test pattern for the next configuration and if the configurations of the CLB are unchanged, the cluster output will remain identical in both configurations. This is helpful because any mismatch between cluster outputs in these two consecutive configurations will indicate the faulty feedback path. Fault diagnosis procedure for the cluster with DFs follows the same steps as in basic cluster diagnosis which involves the observation of ORA cluster output sequence and fault mapping between BUT configuration bitstream and fault list. Details on fault mapping are given in the next chapter.

4.5 Conclusion

Defect tolerant techniques for FPGAs utilize the reconfigurability and allow reconfiguring the application on fault-free resources avoiding the faulty ones. It requires redundant resources which ultimately impacts the testability of the FPGA. The scheme used for hardening the logic blocks increases the fault masking ability of the FPGA thus reducing the fault coverage even in the case of an exhaustive testing. In this chapter, we developed the test schemes for the FPGA blocks hardened with different defect tolerant techniques. The proposed methodologies of testing are extensions of the initial test strategies presented in previous chapter. Some extra configurations are needed to test the added resources. A quantitative analysis of the test cost for defect-tolerant FPGA blocks discussed in this chapter will be presented in chapter 6.

Chapter 5

Test automation and integration into FPGA CAD flow

When an FPGA enters a test mode in order to detect and locate faults, several test configurations are applied. These test configurations must be consistent with the FPGA architectural specifications such as the array size, cluster size, number of cluster inputs/outputs etc. Considering the complexity of the FPGA architecture and targeted fault models, it takes a long time to develop the test scheme and then generate the dedicated test configurations which in turn increases test cost or repair time. The easy way to reduce such cost is the development of test configurations in advance. As mentioned earlier, FPGA logic and interconnect resources are tested separately using dedicated sets of configurations. Therefore, it is preferred to produce sets of test configurations where each set target a specific module/block in the FPGA.

Although testing starts with the development of test algorithm, there is a requirement to translate these algorithms into executable codes that can be used to generate the test configuration bit streams. The configuration bit streams to be loaded into the FPGA are usually generated through a standard CAD flow for FPGAs involving all the necessary steps i.e. synthesis, packing, placement and routing. The challenge is to integrate the test algorithms into the standard flow to produce the required bitstream. Moreover, it is expected to have these codes generalized and scalable such that they can be used for any cluster and FPGA size having the same architectures.

To perform testing by using a BIST, the TPG, ORA and BUT modules should be placed and routed on the FPGA resources as defined in the test scheme. Similarly, signals through clusters and switch boxes will be routed according to the test schemes as well. Traditional FPGA CAD tools do not allow such wide control over manipulating the FPGA resources, which results in the design mapped on the FPGA differently than what was intended by the test engineer. In these tools, modules placement and routing is done by a classical algorithm which is optimized for utilizing lesser FPGA area and minimum

critical path length. In this way, it becomes almost impossible to test all interconnect or logic blocks exhaustively by just utilizing default place and route algorithms. Hence, intrusion into the default placer and router in the FPGA flow is inevitable to perform exhaustive testing which is necessary for higher fault coverage.

5.1 BIST configuration flow

To implement BIST for a specific architecture, there are three main tasks to be performed:

1) Develop BIST module design applications which define the functionality of the BIST modules (i.e. TPG, ORA and BUT).
2) Place the BIST modules on the FPGA clusters defined in the BIST structure.
3) Route the test signals on the dedicated paths defined in the BIST algorithm.

VTR Project is a broad collaboration of researchers which provides a sophisticated environment to perform all necessary steps we need for a given FPGA. Therefore, we employ the basic VTR flow and modify it to adjust our requirements: i.e. produce the configuration bitstream for specific design modules and specific placement and routing. The VTR flow used in this work is shown in Figure 5.1.

As seen in this figure, the flow starts with the description of BIST modules in Verilog HDL format. In our case, n-bit LFSR and n-bit comparator are designed for the TPGs and ORAs respectively. Similarly, LUTs of BUTs are configured to have the functionality of XOR/XNOR for CLB testing or as transparent blocks while testing the crossbar in the cluster.

ODIN-II converts the Verilog HDL design applications into a flattened netlist (*.blif*) which contains logic gates. In the next step, technology independent logic optimization is performed using a tool named ABC and each design is technology mapped onto the LUTs and flip-flops. ABC is a tool that performs synthesis and verification of binary sequential logic circuits in synchronous hardware designs [Betz 1999]. The output file of ABC (*.blif*) is then fed into VPR - a tool flow developed by Toronto University [VPRTO]. The VPR is composed of three main tools. 1) T-VPack, 2) Placer, and 3) Router. The architectural description of the FPGA (*.xml*) is given to VPR.

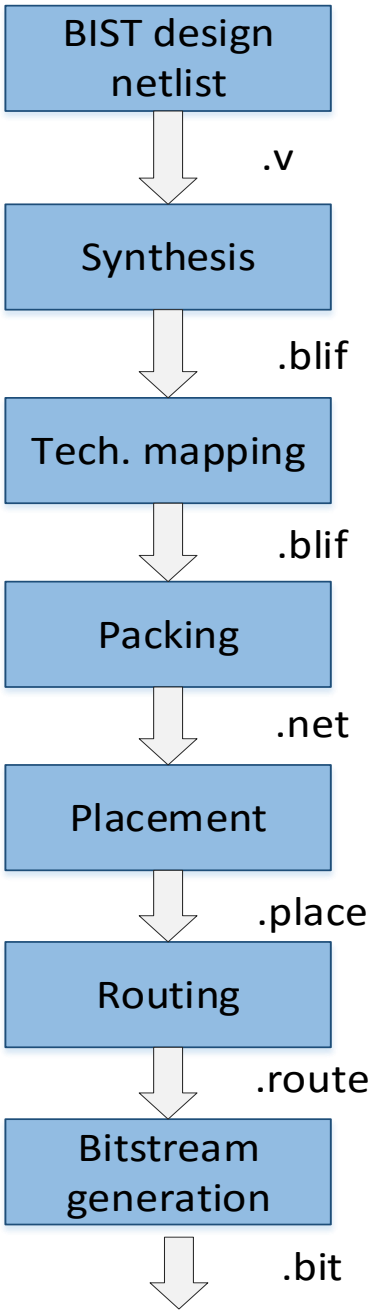


Figure 5.1: VTR Project CAD flow

The function of each tool is described hereafter:

T-VPack: It packs the input netlist into more coarse-grained logic blocks and produces the output file (.net) describing the circuit design for a given FPGA architecture. This step is also called as Bottom-up clustering.

Placer: It performs cluster placement in which CLBs and I/O instances of the application netlist are places on the CLBs and I/O blocks of the FPGA.

Router: The connections between CLBs in the application netlist are established by the Router using the interconnect resources of the FPGA.

Specific placement and routing constraints are given to the Placer and Router in the form of files (*.desc*). These constraint files describe the placement axes and routing nets for the intended BIST structure. The constraints files are developed in a specific format and the following sections describe their development and implementation process as well as their integration into the VTR flow.

5.1.1 Placement constrain files

The VPR's Placer tool uses simulated annealing algorithm as a default placement strategies for any application on FPGA blocks [VPRTO]. This algorithm calculates the ongoing variations of several parameters such as temperature, timing etc. As the placement progresses, it gives an optimal placement with respect to thermal and performance criterion. Although the default algorithm is superior to any user-specified algorithm, the objective here is not to achieve the most optimized placement, but a testability customized placement in accordance with the BIST scheme we developed in chapter 3.

In order to incorporate the testability constrained placement, the default placement algorithm is modified by a method which enables the Placer to read the external instructions and place the design on those clusters specified in these instructions. Usually clusters and switch boxes are assigned on horizontal and vertical axes indicating their positions in the FPGA grid. Therefore, these indices will be also used in the instructions to specify the cluster position.

The format of the placement file we use is given in Figure 5.2 where an example of BUT application placed in the cluster at (1,1) and (2,1) is shown. In the file, column 1 and 2 defines the name and type of the block in the cluster respectively. For example, CLB0-c1 denotes the first CLB (i.e. clb) in the cluster named c1. Third column in the file shows the index of the cluster c1 (e.g. at position (1,1)) and the last column represents the serial number of the CLBs in that cluster.

```
# for cluster 1 1, 2 1, ...
CLB0_c1   clb     1 1     0
CLB1_c1   clb     11      1
CLB2_c1   clb     11      2
.
.
.
CLB6_c4   clb     2  1    6
CLB7_c4   clb     2  1    7
CLB8_c4   clb     2  1    8
```

Figure 5.2: Placement constraints file sample

It is important to mention that if the constraints do not allow to produce a place-able design, the placement process simply aborts giving the indication of invalid placement constraints. Placement process completes only when the design or application is place-able meaning meeting all valid placement constraints. Moreover, the modified placement algorithm can be useful not only in the application of BIST but also for bypassing the defective resources and implementing the application on healthy resources.

When a new BIST structure is formed by placing the TPG, ORA and BUT at their respective places, a number of test configurations are applied keeping the same placement. These configurations are actually based on the different routing constraints which are produced according to the BIST algorithm.

5.1.2 Routing constraints files

After the placement of BIST modules, connection among logic blocks and interconnect structures is established. These connections must respect the conditions defined in the BIST algorithm as well. Therefore, routing constraints are given to the router to define the intended paths signals that should be propagated though it during the test. These constraints are not only for the crossbars in the cluster and switch box but also for the CLBs in the cluster.

VPR's routing tool is based on Dijkstra's Path finder algorithm. This algorithm starts with the definition of the routes without taking into account the usage of interconnect and

logic resources. Once the routability of the given application is established, rerouting is performed in iterations to find the shortest possible path. The criterion observed in finding the shortest path is to reduce the over usage of a routing resource. This is done by forcing signals to the alternate nets with less usage, leaving behind only the net that needs a given fully specified resource [VPRTO].

The routing algorithm in VPR tool is modified to read the routing constraints given in the form of a file. It is important to mention that modifications have been made to consider both types of files 1) those files having the list of 'must be used' blocks/nets and 2) other files having the list of 'do not use' blocks/nets. Therefore, two versions of Router algorithm exist, each accepting the corresponding type of file. It means that the test engineer has the flexibility to implement the BIST algorithm for producing the routing constraints either listing 'must be used' blocks/nets or 'do not use' block/nets. The usage of either version of router in the flow is managed by the parameters given in the command. In these files, the complete hierarchy of the blocks/nets is mentioned which also simplifies the implementation of the BIST algorithm.

```
# for cluster 1 1

Cluster    1 1    dmsb  1
Cluster    1 1    dmsb  2
Cluster    1 1    dmsb  3
Cluster    1 1    dmsb  0      mux   1
Cluster    1 1    dmsb  0      mux   11
Cluster    1 1    dmsb  0      mux   21
Cluster    1 1    dmsb  0      mux   31
Cluster    1 1    dmsb  0      mux   41
```

Figure 5.3: Routing constraints file sample

To illustrate further, an example of such a file is shown in Figure 5.3 which lists 'do not use' blocks/nets. It starts with providing the axis of the main block i.e. cluster and switch box in the FPGA mesh. Then it mentions the name and number of the crossbar i.e. DMSB 0 or UMSB 0 which are indexed in their main block. The lowest granularity of the instances that can be used here is the MUX (2:1) which is mentioned also with its name and index. As it is a hierarchical description, the instance needed to be bypassed or

avoided is mentioned with its complete hierarchy. For example, Figure 5.3 shows the scenario of testing the first DMSB in the cluster where other three DMSBS are not used. The first three lines in the routing constraint file depict this condition to avoid DMSB 1, 2, and 3 completely. Furthermore, in the first DMSB of cluster at axis (1,2) MUX 1, 11, 21, 31 and 41 are also avoided forcing the router to use other MUXs in DMSB 0 to propagate the signals.

Every new test configuration corresponds to a new routing constraint. The validity of each routing constraint is established when the Router successfully routes the design under the given constraints. If any of the placement and routing constraint is found to be inapplicable, the new routing or placement constraints are produced based on the developed BIST algorithm and tried for a valid placement and routing.

5.1.3 Integration into CAD flow

BIST schemes developed in this work are scalable to any FPGA array size. As shown in chapter 3 the BIST scheme developed for switch box is implemented on 2x2 FPGA resources and can be extended to have *N* parallel 2x2 BIST for *NxN* FPGA array size. Similarly, the developed algorithms are applicable to a range of cluster sizes. Therefore, to simplify the implementation of proposed schemes and integrate them into the standard CAD flow, we develop sets of tools to design the BIST architecture and produce the constraints files. Moreover, the developed tools are fully automated and generalized to consider any cluster and any FPGA array size given by the user/test engineer.

To perform the generalization of the developed tools, we need to consider variables in the FPGA architectural description given to the VPR in the CAD flow. Figure 5.4 summarizes the architectural parameters to be inserted in the design and passed to the T-VPack, Placer and Router to deal with their respective step. The architectural parameters given at each step are as follows:

Bottom-up clustering step: During clustering architectural constraints are imposed i.e. cluster size, the number of a CLB inputs/outputs and the number of a cluster inputs/outputs.

Clusters placement step: At this step, the only parameter given to the Placer is the array size of the targeted FPGA.

Routing step: At this step, the architecture's Rent parameter is given to the Router. This parameter is selected such that it gives the optimal architecture size for the particular

cluster size parameter given at the clustering step. Moreover, channel width is also selected in order to give a routable design.

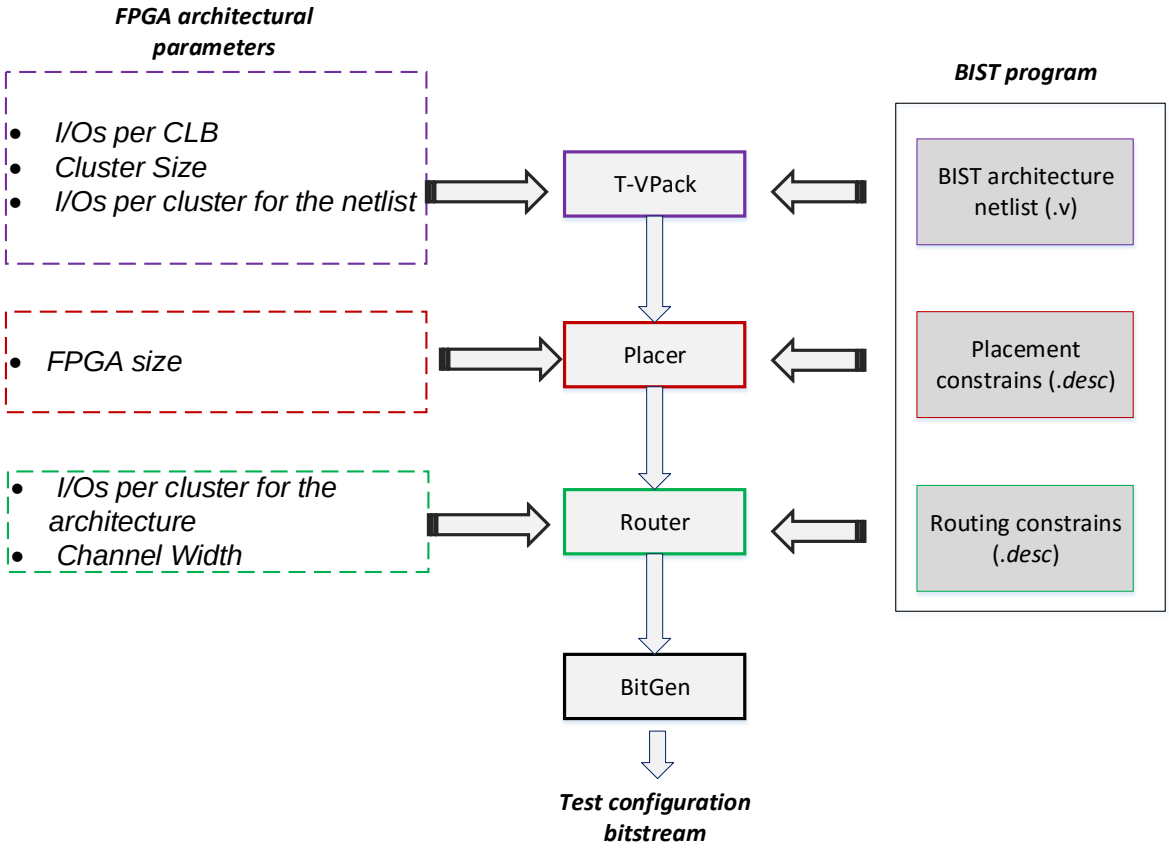


Figure 5.4: BIST integrated into FPGA CAD flow

As mentioned in chapter 2, the number of cluster inputs/outputs is chosen with the help of Rent's parameter. This parameter gives an optimized architecture in terms of locality of the logic computation. In this context, we can define Rent's parameters for FPGA architecture netlist and application netlist. The netlist Rent's parameter depends on the input/output numbers of a cluster in the netlist. This inputs number is used to group CLBs in a cluster. If the netlist Rent's parameter decreases, the cluster inputs/outputs number decreases and by consequence the number of clusters of the netlist increases [Pistorius 2003]. The relationship between the two Rent's parameters is established depending on the architecture's type of interconnect (full or depopulated crossbar). The architecture's Rent parameter is greater than the application's Rent parameter in the case of the fully populated crossbar interconnect whereas the two Rent parameters are equal in case of depopulated crossbar interconnect.

The tools developed to generate placement and routing constraints can work with any cluster size, FPGA array size and block under test size, thus being scalable. This ensures

the compatibility and consistency with the parameters given to *T-VPack* and other tools by the same user.

5.2 BIST Validation flow

To validate the developed BIST schemes, we need to perform a qualitative analysis. This analysis must evaluate the efficiency of our schemes in terms of testability. Usually testability of a scheme is calculated in terms of the fault coverage defined as the ratio between detected faults and the total number of possible positional faults in a given circuit. In the case of FPGAs as for any other ASIC circuit, the fault coverage is linked with the test cost. In the case of FPGA BIST the test cost is related to the test time. The dominant factor in FPGA test time is the time required to load a configuration. Therefore the goal is to minimize the number of test configurations to achieve certain fault coverage.

The typical way to validate a test scheme is to inject faults and perform test simulation to get the respective fault coverage. The CAD flow used in the previous section for the bitstream generation does not have the capability to perform such kind of verification. Therefore, we need to find a way to inject faults when the BIST architecture is mapped to the FPGA and perform the test simulation. Since we are targeting stuck-at and bridging faults, we need to inject them in the gate-level representation of the FPGA. Thanks to the project partners working on the development of the FPGA architecture generator, we could have access to the complete FPGA architecture netlist and we were able to use it according to our needs. For this reason, we employ the standard tools used for ASICs such as *Design Vision* and *TetraMAX* from *Synopsys*, for FPGA netlist synthesis and injecting faults in the FPGA architecture and evaluate the test and fault coverage using typical CAD flow. In the following, the details of the verification methodology adopted in this work are given.

5.3 Validation methodology

The developed BIST schemes for cluster and switch box are validated to quantify their efficiencies in terms of number of test configurations and respective fault coverage. It involves the injection of faults at the targeted module and then applying BIST configurations to detect those faults. It requires the development of tools that can translate the BIST configurations into the constraints that can be applied in this validation methodology. The details of this methodology are as under.

The BIST validation starts with performing some minor modifications in the VHDL netlist. It is because we are using synthesis and ATPG tools (i.e. *Design Vision* and *TetraMAX*) available for ASICs. Starting with the cluster, separate primary input pins (i.e.

ram [x:0]) are added to the cluster netlist which are directly connected to the 'select' pins of the multiplexers, emulating the SRAM signal to the MUX. In this way, multiplexers can be controlled/configured by the logic values at the dedicated primary inputs. The structure of a cluster with added pins (ram[x:0]) is shown in Figure 5.5.

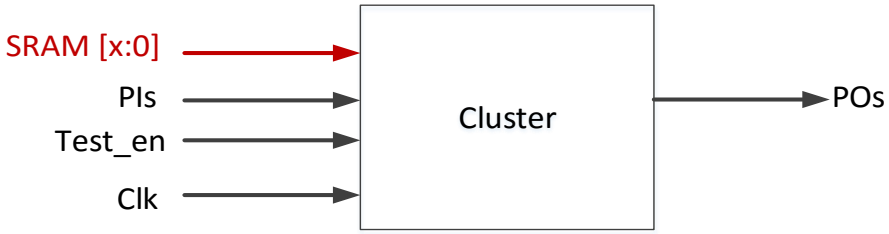


Figure 5.5: Cluster with SRAM [x:0] as primary inputs

Considering the configurations developed for cluster testing, where each MUX (2:1) is configured with a single bit, selecting one or the other MUX inputs, we apply the exact bit at the dedicated primary inputs (ram [o:x]) added to the cluster block. Using these inputs, we are able to select test paths as in the case of FPGA configuration flow. To illustrate this idea in more details, we consider an example of crossbar testing.

In the case of an exhaustive testing of a crossbar, all paths should be activated one by one. Figure 5.6 shows the structure of a 11:1 MUX made of several 2:1 MUXes. This complex MUX can be considered as one of the *N* identical MUXes in a single crossbar where *N* is the cluster size. Considering the hierarchical structure of the crossbar, 11:1 MUX is made of 4 levels. At each level, all MUXes have a common select signal (i.e. Sel 0 at level 1). These select signals are then connected to the ram [x:0] inputs which are added as primary inputs of the cluster. In this way, all paths in MUX 11:1 can be activated one by one by applying input constraints at ram [x:0] signals of the cluster.

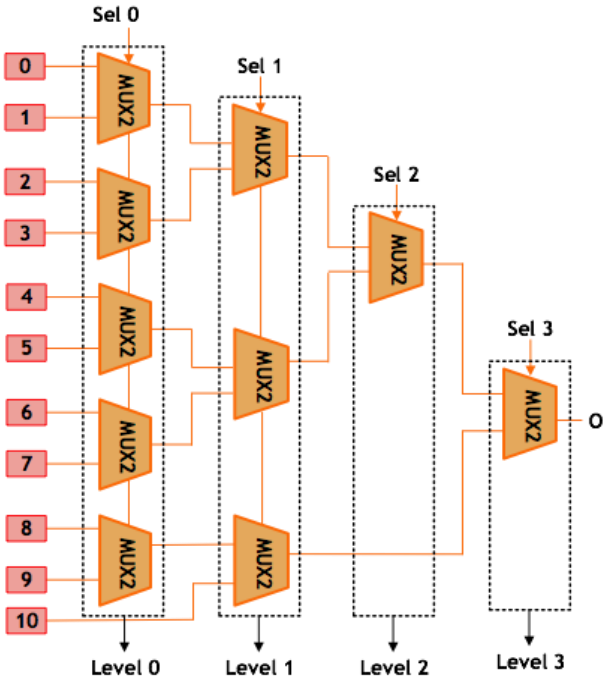


Figure 5.6: 11:1 MUX as hierarchical structure made of 2:1 MUXes

As discussed in chapter 3, LUTs are configured as transparent blocks during the testing of cluster crossbars. For a 4 input LUT, a single input is propagated through the corresponding crossbar 'Down' during the test steps. For example, for testing first crossbar 'Down', input A of the LUT is propagated to the CLB output and eventually to the cluster output. Similarly, input B of the LUT is propagated to the CLB output for second crossbar down testing and so on. The input signals 'SRAM [15:0]' of LUT MUXes are connected to cluster input ram [x:0] so that the dedicated input signals can be applied to emulate the configuration of a MUX. The LUT SRAM [15:0] configurations to propagate input signal A, B, C or D to LUT output are shown in Table 5.1.

After the application of these input configuration signals according to the test scheme, stuck-at or bridging faults are injected at all potential nodes in the cluster architecture. For an exhaustive testing of a block, many fault simulations have to be performed for a given input for SRAMs according the BIST scheme. To do that, we develop an automated flow which is explained below.

TABLE5.1: LUT CONFIGURATION BITS FOR PROPAGATING ANY OF ITS INPUTS TO THE OUTPUT

The LUT SRAM configuration	Propagating A	Propagating B	Propagating C	Propagating D
SRAM[0]	0	0	0	0
SRAM[1]	1	0	0	0
SRAM[2]	0	1	0	0
SRAM[3]	1	1	0	0
SRAM[4]	0	0	1	0
SRAM[5]	1	0	1	0
SRAM[6]	0	1	1	0
SRAM[7]	1	1	1	0
SRAM[8]	0	0	0	1
SRAM[9]	1	0	0	1
SRAM[10]	0	1	0	1
SRAM[11]	1	1	0	1
SRAM[12]	0	0	1	1
SRAM[13]	1	0	1	1
SRAM[14]	0	1	1	1
SRAM[15]	1	1	1	1

5.4 Validation flow

The verification flow starts with the synthesis of the targeted block netlist (i.e. cluster or switch box) having additional input pins for SRAMs. The first step is the design elaboration. The elaboration phase performs a generic pre-synthesis of the analyzed models. It essentially identifies the registers that will be inferred. Then design environment is defined which includes operating conditions, wire load models and system interface characteristics.

Next is the compilation phase which performs the assignment of logic gates from the standard cell library to the generic gates in the elaborated design in such a way the defined constraints are met. When the design compilation is done, synthesized netlist of the FPGA is generated.

Faults are injected at every potential node in the synthesized netlist of the FPGA. Then comes a critical step in which input constraints developed for emulating BIST configurations are applied.

After performing design rule check (DRC), test patterns are applied from an external file containing the exact same test patterns produced for BIST scheme. At the end of each simulation, fault and test coverage is observed. The number of fault simulations corresponds to the number of BIST configurations. Once all fault simulations of a block are performed, cumulative fault coverage is evaluated where each configuration contributes to a subset of injected faults detection. If a simulation does not increase the fault coverage (either due to its inability to detect any fault or it detects the faults already detected in the previous simulations), that simulation hence the test configuration is discarded and is not considered in the BIST configuration. If the required fault coverage is not achieved, new configurations are applied and this iteration continues until the 100% fault coverage is obtained. The faults detected from these configurations are conserved in a list, where each fault is designated with the affected instance name and number (according to the nomenclature given in the synthesized gate-level netlist). In this way, a particular configuration targets a specific subset of faults. This information is used at the end of the flow to extract the location of the fault. The flow used for BIST verification is summarized in Figure 5.7.

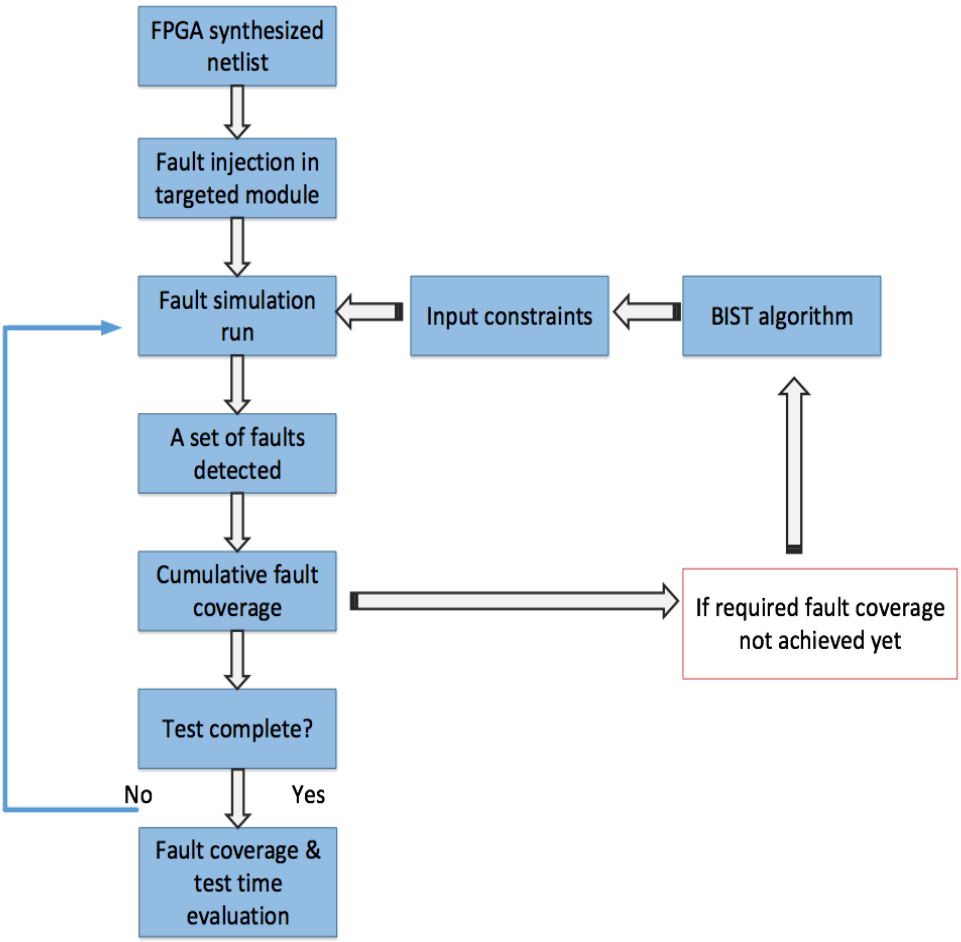


Figure 5.7: BIST algorithm verification flow

To translate the BIST algorithm into applicable input signals values for SRAM, a set of programs is developed. These programs develop the script (*.tcl*) for the fault simulator containing the SRAM signals as primary input constraints. For each BIST configuration, there is an equivalent script for fault simulator. These programs are generic and produce the required number of scripts containing primary input constraints according to the block under test and the size of the cluster given by the user.

5.5 Fault diagnosis

According to the BIST schemes, faults in the FPGA logic blocks and interconnect are detected by observing mismatch at the ORA outputs. However, finding the exact location of the detected fault is not straightforward. It is due to the fact that testing a particular block in the FPGA involves configuring the complex hierarchical structures which are not currently under test but involved in the propagation of the test inputs to specific outputs. Diagnostic resolution is very critical in this context that can be defined as the hierarchical

level of the FPGA instances through which a fault can be traced back. High resolution means that a fault can be diagnosed at the lowest instance i.e. MUX or gate level. It is obvious that for higher diagnostic resolution, the number of required configurations and thus the test cost will increase.

In our case, we target the diagnostic resolution of MUX level (2:1) which is the lowest hierarchical instance both in the logic block (LUT) and the interconnect (crossbar). However, we are able to achieve even higher diagnostic resolution (at gate level) within a 2:1 MUX, thanks to the mapping mechanism we developed using BIST configuration and verification flows. The details of the mapping mechanism are as follows.

5.6 Fault mapping

Observing the ORA output as '1' or '0' does not explicitly give the information about the location of the fault. However, the sequence of the ORA cluster outputs (i.e. O_1, O_2...O_n) is very important in detecting the fault type and the faulty path. For instance, in case of stuck-at fault detection, such test patterns are applied which produce identical outputs on every cluster output. Therefore, any mismatch among two BUT cluster outputs will identify stuck-at fault and the faulty path will be located by observing the output sequence (e.g. O_2 is coming through second MUX in crossbar 'Up' which is connecting second CLB and so on...). As a single BUT is compared in two different ORAs, fault masking problem can easily be overcome. And in case of bridging fault detection, test patterns are applied producing alternate logic values at the cluster outputs. That means O_1, O_2, O_3,...O_n gives 1, 0, 1,....0. Therefore, a non-mismatch between two consecutive BUT cluster outputs will indicate a faulty path.

For each BIST configuration developed using BIST configuration flow, there is an equivalent script obtained by using verification flow (given in section 5.4). For each script, we associate a set of faults that can be detected in that particular BIST configuration. Therefore, a direct mapping can be established between the BIST configuration and the list of potential faults that can be detected using this configuration.

For further illustration, mapping of a bitstream and fault list is shown in Figure 5.8. On the left, there is a bitstream (*.bit*) generated by using configuration flow and on the right is the fault list generated by the verification flow using the same configurations. As can be seen here the (*.bit*) file is very elaborative containing the axis of the blocks (e.g. CLB) along with the configuration bits of each MUX in that CLB. Similarly, fault list contains the type of faults (sa0- stuck at '0'; sa1- stuck at '1') and the complete hierarchy of this fault (e.g. crossbar type and name, then the MUX name and the gate).

When a fault is detected at the ORA cluster output, that particular faulty output can be traced back giving the faulty path passing through several instances of the BUT cluster/switch box. The current BIST configuration of the BUT cluster/switch box gives automatically the full potential fault list of this complete path. Fault could exist in any of the instances involved in the faulty path. However, to establish a starting point while testing, it is assumed that the fault exists in the instance which is currently under test (e.g. crossbar 'Down' of a cluster). This assumption can be validated because other instances involved in the very path (e.g. CLB and crossbar 'Up' of a cluster) will be tested exhaustively during their respective test phases. We know that during an exhaustive testing of an instance, all possible path combinations are tested one by one keeping the configuration of other involved instances unchanged. In this way, if the assumption about faulty instance we made earlier is found to be wrong, similar faulty outputs will be observed for all path combinations throughout the other instance testing.

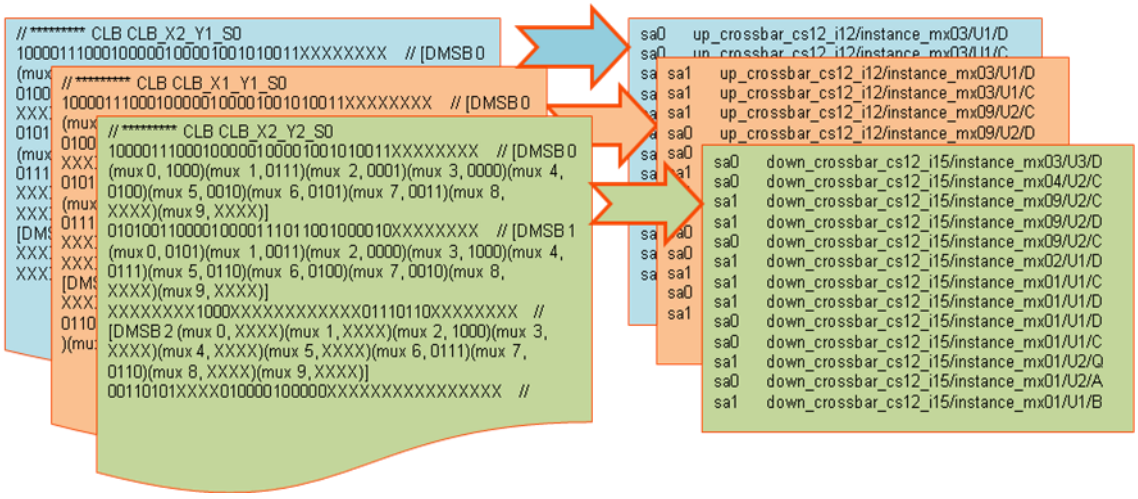


Figure 5.8: Fault mapping between configuration bitstreams and fault lists

As the fault list contains the hierarchy of all the instances and the corresponding faults, simple manipulation by tracing back in the faulty instance gives the possible faulty MUX input or even gate where fault exists.

Consider an example where an ORA cluster of size 12 has an output (O_1, O_2, O_3...O_{12}) as (00 00 10 00 00 00) while testing 'first' of the four crossbars 'Down' for stuck-at faults in a cluster at (2, 2) of an FPGA array. The output O_5 of ORA cluster indicates a stuck-at fault. The current BIST configuration of the cluster suggests that this fault exists in a path that can be traced back through fifth CLB to the fifth MUX (11:1) of the 'first' crossbar 'Down' in the BUT cluster. Hence the fault exists in the fifth MUX (11:1) of the 'first' crossbar 'Down' in BUT cluster. As mentioned earlier, every MUX is tested for its all paths one by one; the only active path in the fifth MUX (11:1) of the 'first' crossbar

'Down' will give the faulty MUX (2:1). From the fault list corresponding to this particular BIST configuration, the faulty pin (input/output) of the MUX (2:1) and the type of fault (sa0/1) is established.

5.7 Conclusion

In this chapter we have presented some CAD tools that have been developed for BIST implementation, validation and integration into the standard CAD flows. VPR tools for FPGAs provide a complete flow including synthesis, packing, placement, routing and bitstream generation. We employed these tools and used them according to our needs. It requires development of some set of tools. Our contribution is summarized as follows:

- Several tools have been developed to generate BIST designs/modules that perform the functions of TPG, ORA and BUT. As the number of test patterns to be developed depends on the number of paths under test, TPG/ORA and BUT tools need to be generic where size of the BUT is given by the user/test engineer.
- Several tools have been developed to produce place and routing constraints to be provided to the existing Placer and Router tools in VPR for BIST bitstream generation. The set of programs that translates the proposed BIST scheme into the constraints files is generic and can be used for any FPGA array size, cluster size, given by the user.
- Some tools have been developed to translate the BIST scheme into the .tcl scripts used for BIST scheme verification in TetraMAX (Synopsys). At this step of validation flow, each BIST configuration is emulated and the fault simulation is performed. The set of programs that constitute the tool are generic and produce the .tcl scripts for a given block to be tested, or cluster size and the number of cluster I/Os.
- Several tools have been developed to generate cumulative fault coverage reports once the fault lists and block's individual fault coverage corresponding to each BIST configuration are obtained. These tools map every BIST configuration to its fault list and sort them in the order of fault coverage.

These tools are generated using Perl and C++.

Chapter 6

Experimental evaluation and results

Testability is one of the most important attributes involved in the reconfiguration operations of a given FPGA architecture when detecting and locating defects. Testability is also important if we want to scale the efficiency and performance of the related defect-tolerant schemes. At the bottom line of any FPGA based fault tolerance technique, we found the defect types and their location especially for those schemes which rely on defect bypass mechanism. Testability is calculated in terms of fault coverage which is defined as a ratio between detected faults and the total number of potential faults in a given circuit. The other metrics of testability include testing time and the corresponding fault coverage for a given test vector.

As explained in the previous chapter, an FPGA array is composed of two basic building blocks. i) Cluster, and ii) Switch box. For the particular architecture of FPGA considered in this thesis, we have developed different BIST structures for the cluster and switch box. In the 'BIST for cluster' structure, we have presented test configurations for all the components in the cluster. Similarly 'BIST for switch box' tests all the components of the switch box. In this chapter, a qualitative analysis of the proposed BIST schemes is presented. Results are obtained by performing fault simulations according to the flow presented in chapter 5, implemented on the proposed BIST schemes. Moreover, the performance of the BIST scheme is compared with the standard scan-DFT results.

6.1 Impact of cluster-size on the FPGA testability

This thesis is a part of the project that deals with the development of the robust FPGA architecture. For that matter, we have the freedom to explore the architectural parameters impacting the testability and the test cost of the resulting FPGA. We implement the proposed BIST schemes to evaluate their performance on a number of FPGA architectures having different cluster sizes, hence different number of interconnect in the cluster and switch box. The details for cluster size optimization for a better area vs. routability are given in chapter 3. Table 6.1 gives an overview of the number of cluster

inputs and outputs for different cluster sizes which are explored here. For all the cluster sizes, the number of crossbar Down' and crossbar 'Up' in a cluster remains the same being 4 and 1 respectively. The number of inputs of a crossbar 'Up' is equal to the cluster size whereas the crossbar 'Up' or the cluster outputs changes with the cluster size. The column four in Table 6.1 gives the number of inputs of a single crossbar 'Down' comprising the cluster inputs and feedbacks from the CLBs. The cluster inputs and feedbacks are uniformly distributed among crossbar 'Down'. Each CLB has its outputs at the cluster outputs; therefore the number of feedbacks is also equal to the cluster size.

TABLE 6.1 CLUSTER I/OS FOR DIFFERENT SIZES

Cluster Size	**Number of cluster inputs**	**Number of cluster outputs**	**Number of inputs in a crossbar 'Down' (cluster inputs + feedback)**
4	12	4	3+1
6	16	8	4+2
8	20	8	5+2
10	24	12	6+3
12	28	12	7+3

It is evident that changing the cluster size changes the number as well as the size of MUXes in the crossbars. For instance, a single crossbar 'Down' is composed of 8 MUX 7:1 and 12 MUX 10:1 in the cluster of size 8 and 12 respectively.

In case of switch box, the number of DMSBs and UMSBs depends on the number of cluster inputs and outputs respectively. From the FPGA architecture we know that;

$$No.of\ UMSBs\ in\ switch\ box = \frac{No.\ of\ cluster\ outputs}{4} \quad \text{(eq. 6.1)}$$

$$No.of\ DMSBs\ at\ 'Level'\ of\ switch\ box = \frac{No.\ of\ cluster\ inputs}{4} \quad \text{(eq. 6.2)}$$

$$No.of\ DMSBs\ at\ 'Level\ 2'\ of\ switch\ box = \frac{channel\ width}{2} \quad \text{(eq. 6.3)}$$

Table 6.2 presents the summary of the number of UMSBs and DMSBs at 'Level 1' and 'Level 2' of the switch box for various cluster sizes. It also shows the minimal channel

width required to route all the MCNC benchmark circuits [MCNC] in case of each cluster size. As explained in earlier chapters, channel width is defined by the number of wires connecting two switch boxes together. It defines the routability of the FPGA architecture and is selected by simulating the benchmark circuits for their routability. The details of 20 MCNC benchmark circuits and the area required to implement and route each of them on the given FPGA size is presented in Appendix A.

TABLE 6.2 UMSBS AND DMSBS IN A SWITCH BOX FOR DIFFERENT SIZES

Cluster Size	Number of UMSBs	Number of DMSBs at 'Level 1'	Number of DMSBs at 'Level 2'	Channel width
4	1	3	15	30
6	2	4	17	34
8	2	5	23	46
10	3	6	23	46
12	3	7	29	58

As explained in the previous chapters, the proposed BIST schemes perform exhaustive testing for all interconnect. Therefore we can expect the direct impact of cluster size on the testability and test cost of the cluster and switch box which is presented in the following sections.

6.2 BIST simulation results

To quantify our analysis, BIST architectures are implemented for various cluster sizes and the test cost is determined for both the cluster and the switch box. BIST fault simulation is performed using the BIST verification CAD flow described in the chapter 5. The main steps in BIST simulations involve the configuration of the FPGA blocks according to the BIST scheme and then fault injection at the targeted module. For that purpose, we employ *Synopsys TetraMAX®* - a reliable vector generation and fault simulation tool and a high performance platform for testing methodologies.

6.2.1 Results of BIST implementation for cluster

There are three types of blocks in the cluster. 1) CLBs, 2) Crossbar 'Up' and 3) Crossbar 'Down'. The algorithm proposed for the cluster BIST starts with testing of CLBs, then crossbar 'Up' and ends with the testing of all the crossbars 'Down' taking one at a time.

For simulation purposes, stuck-at and bridging faults are injected into the gate level netlist of the cluster. TetraMAX® tool produces exhaustive fault lists for all potential faulty nodes in the cluster, targeting standard stuck-at and bridging fault models. For each block in the cluster, a certain number of test configurations developed using BIST schemes are applied. To do that automated *.tcl* scripts were generated as described in chapter 5. For each test configuration, test patterns are applied which detect a set of injected faults. And at the end of each simulation, fault coverage is evaluated.

Table 6.3 shows the results produced in the form of the number of configurations required to test each block in the cluster to achieve 100% fault coverage. Since all CLBs in the cluster are tested simultaneously irrespective of cluster size, the number of test configurations for CLBs remains 4 for all cluster sizes. Second column of the table shows the number of test configurations required for sequential testing of all four crossbars 'Down' in a cluster. Parallel testing of crossbars 'Down' reduces the diagnostic resolution as explained in chapter 3. Therefore, all crossbars 'Down' are tested sequentially - one after the other. And this is why; the major portion of the number of test configurations is contributed by crossbars 'Down'. The total number of test configurations required for a single cluster is given in column five of the Table 6.3.

TABLE 6.3 RESULTS FOR CLUSTER BIST

Cluster Size	**Number of configurations required to fully test & diagnosis a cluster**				**Test Cycles #**
	CLB	**Crossbars 'down'**	**Crossbar 'Up'**	**Total**	
4	4	16	4	24	378
6	4	24	11	39	1065
8	4	32	12	48	1664
10	4	58	16	78	3875
12	4	60	18	82	4770

Considering an example of Xilinx FPGAs, configuration bitstreams are loaded into the device through special configuration pins. These pins serve as an interface for different configuration modes. Mostly *SelectMAP* master/slave configuration mode is used which supports either serial or parallel data path. The *SelectMAP* interface provides 8-bit, 6-bit and 32-bit bidirectional data bus for configuration. Generally, master mode is a self-loading FPGA configuration mode. In this mode, configuration bitstreams are

stored in non-volatile memory on the same board and external to the FPGA. A configuration clock signal is generated by the FPGA which drives the configuration logic. Thus, it is the FPGA that controls the configuration process. In slave mode, configuration loading is externally controlled either by a processor, microcontroller or tester. In this mode, clock signal is an input signal. The advantage of slave mode is the flexibility to store configuration bitstream anywhere in the system [XilinxRpt]. At slave *SelectMAP* interface, configuration data loading is controlled by three signals;

1. Chip select input (CSI_B) which enables SelectMAP bus.
2. Read/write input (RD/WR_B) which controls whether data pins are inputs or outputs.
3. Clock input (CCLK), FPGA samples the data pins at the rising edge of clock signal.

In this work, we consider a *SelectMAP* slave like interface; a single bus of width 8-bit for loading configuration data onto the SRAM cells of the cluster. The last column of Table 6.3 presents the number of cycles required to configure the cluster for test procedure.

Figure 6.1 shows the plots of the attained fault coverage for each cluster size. It is deduced from the plots that 80% of the fault coverage is achieved by 35-40% of the total test configurations. It is also very significant that most of the faults are covered with initial test configurations which can be helpful in the situations with short test time availability when applying fewer test configurations will lead shortly to relatively higher fault coverage.

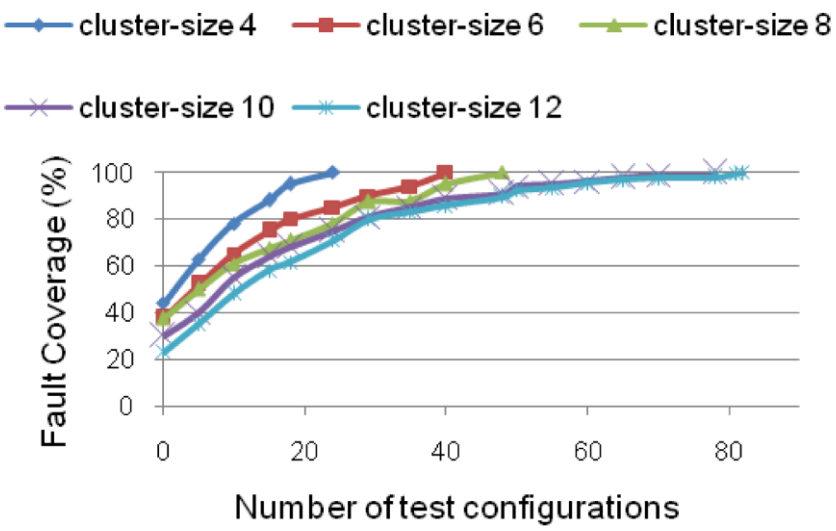


Figure 6.1: Fault coverage vs. Number of test configurations of a cluster

From the results we can conclude that for different cluster sizes, testing of cluster of size 6 and 10 is relatively more expensive as compared to the increment in their cluster sizes from 4 and 8 respectively. For instance, increasing the cluster size from 4 to 6 increases the number of test configuration by 62.5% whereas increasing the cluster size from 6 to 8 requires only 23% more test configurations. Similarly, increase from cluster size 8 to 10 needs 62.5% and from 10 to 12 needs only 1.34% more test configurations. It is due to the architectural constraints where outputs of the cluster and local feedback paths have to be a multiple of 4. Overall, the cluster of size 8 and 12 appear to be the optimum choice if we consider only the cluster test cost.

6.2.2 Results of BIST implementations for switch box

Regarding the switch box architecture, three types of blocks are mentioned in the switch box. i) UMSBs at hierarchical 'Level 1', ii) DMSBs at 'Level 1' and iii) DMSBs at 'Level 2'. Similar to the cluster BIST, the BIST scheme for switch box tests the interconnects in several phases. Simulation is performed by injecting faults at the gate level netlist of the switch box architecture using standard stuck-at and bridging fault models taken into account by *Synopsys TetraMAX®*.

The efficiency of the test scheme is calculated in terms of the number of test configurations required to reach 100% fault coverage. The results for cluster size 4, 6, 8, 10 and 12 are again considered and given in Table 6.4. This table shows the number of test configurations required to attain 100% stuck-at and bridging fault coverage for different switch box modules. All UMSBs at 'Level 1' are tested simultaneously using a set of 5 configurations irrespective of the cluster size. Similarly, all DMSBs at 'Level 1 and 2' are tested simultaneously in their respective test sessions. There is a noticeable difference between the number of test configurations required for DMSBs at 'Level 1 and 2'. The reason is the large number of testable paths in DMSBs at 'Level 2' and a limited number of selectable paths per configuration. The increased number of test configurations for DMSBs at 'level 1' for cluster of size 4 and 6 is due to the architectural parameters which give fewer number of DMSBs, but each DMSB has more inputs compared to a cluster of size 8. For this reason, the number of configurations in case of cluster of size 6 is the same as of size 8.

TABLE 6.4 RESULTS FOR SWITCH BOX BIST

Cluster Size	Number of configurations required to fully test & diagnose a switch box				Test Cycles #
	UMSBs	DMSBs at 'Level 1'	DMSBs at 'Level 2'	Total	
4	5	5	16	26	1740
6	5	6	18	29	1918
8	5	4	20	29	1918
10	5	5	36	46	2560
12	5	8	40	53	2700

It is also important to note that the increment in the cluster size from 8 to 10 costs more test configurations (i.e. ~58%) compared to the increment from size 10 cluster to size 12 (i.e. ~15%). It is due to the fact that cluster of size 10 has some redundant interconnect at its output to retain the architectural symmetry which needs cluster outputs to be a multiple of 4 i.e. 10 CLBs with 12 cluster outputs. The last column of the Table 6.4 presents the number of test cycles required to configure a switch box for the test procedure. We again consider a standard 8-bit wide bus to load data onto the SRAM cells of the MUXes in the switch box.

6.2.3 Test time optimization results using joint testing

A test time optimization technique was proposed in chapter 3 (section 3.6) in which joint testing of 'crossbar up' and CLBs is performed. In brief, CLBs are configured with XOR and XNOR functions while testing 'crossbar up' interconnect. Fault diagnostic is done by manipulating the output sequence of the ORA cluster. In this way, the number of configurations for testing CLBs separately can be avoided; hence the test time required for CLBs testing can be saved. The resulting number of configurations for various cluster size is given in Table 6.5.

A comparison is presented in Figure 6.2 showing the gain achieved for joint testing compared to separate testing of crossbar 'Up' and CLBs. The gain is prominent in lower cluster sizes as expected because the number of configurations saved in joint testing is constant. Therefore, the percentage gain is reduced as the total number of test configuration increases. For example the lowest gain in test time (i.e. 4.5%) is found to be in the case of cluster of size 12.

TABLE 6.5 RESULTS FOR OPTIMIZED TEST CONFIGURATIONS

Cluster Size	Number of Test Configurations Required for 100% Fault Coverage			Test Cycles #
	Crossbar 'Down'	CLB + Crossbar 'Up'	Total	
4	16	4	20	324
6	24	11	35	966
8	32	12	44	1536
10	58	16	74	3686
12	60	18	78	4551

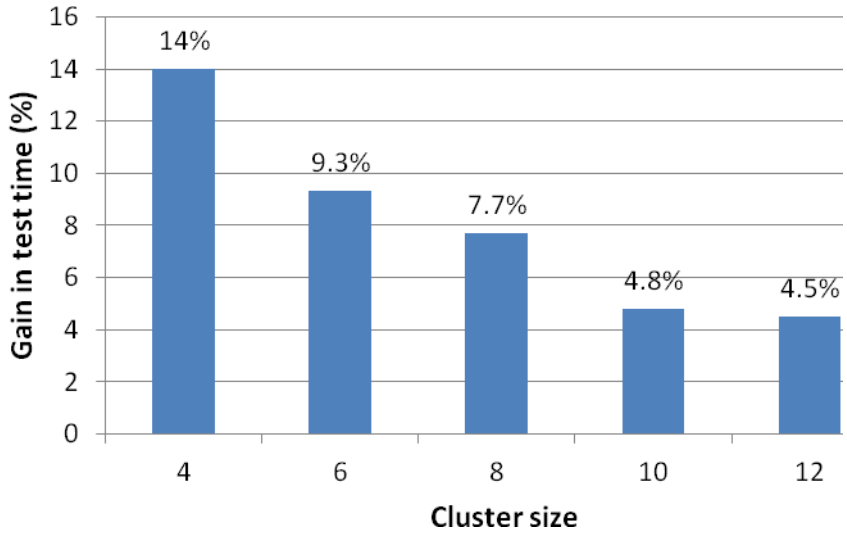


Figure 6.2: Comparison of joint and separate testing of CLB and crossbar 'Up' for various cluster sizes

Another feature related to BIST configurations: the order of configuration sequences is important to achieve higher fault coverage relatively rapidly at the beginning of test configuration generation. Figure 6.3 depicts the plot of fault coverage vs. number of configurations for joint testing.

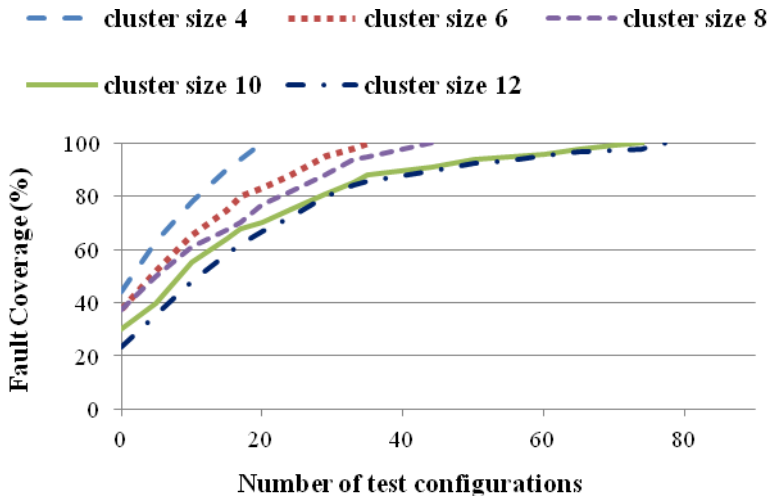


Figure 6.3: Fault coverage vs. number of configurations for joint testing

The optimization results using joint testing technique show that with increasing the cluster size from 4 to 6 requires 20% more test configurations along with the reduction of 5% in test time gain compared to separate testing of crossbar 'Up' and CLBs. On the other side, increasing cluster size from 10 to 12 requires 5% more test configurations with the decrement of only 1% in test time gain.

6.2.4 Test time optimization results using partial reconfiguration

To program an FPGA for an application, dedicated configuration bits are loaded into the FPGA memory array (SRAM cells). These configuration bits are stored outside the chip and grouped into the frames of a certain bit-width, depending on the type of the FPGA considered. For loading an application, these bits are accessed one frame at a time and written into the SRAM cells. Therefore, exhaustive testing becomes very costly for those FPGAs which require full reconfiguration to be loaded each time to implement an application.

Some FPGAs provide the facility to configure some of their modules, keeping the configuration of others unchanged. In this case of partial reconfiguration, only the configuration bits needed to be modified are loaded into the FPGA. Thus, the number of frames required for a new FPGA configuration is reduced. We explore this feature of partial reconfiguration during FPGA testing where test time reduction can be achieved by using less number of frames per configuration.

FPGA configuration bits stored in the external memory are loaded into the FPGA through dedicated port as discussed earlier in this chapter. The scenario is depicted in

Figure 6.4 in which the FPGA is fully configured using 'F-bits'. In case of full reconfiguration, 'F-bits' are loaded each time for a new configuration. In partial reconfiguration, FPGA is fully programmed only for the first configuration. In the following ones, only the configuration bits ('P-bits') required by the under test module (e.g. cluster) are loaded. Thus, the number of frames required in partial reconfigurations is reduced.

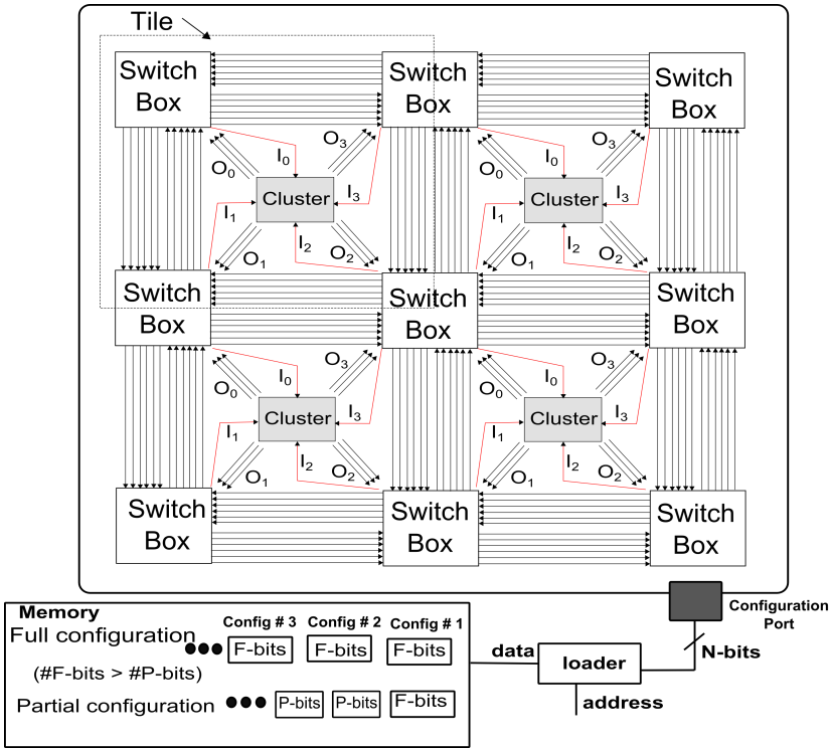


Figure 6.4: Mesh of clusters FPGA with configuration port

The BIST structure and the implementation for partial reconfiguration remains the same as in the case of full reconfiguration. Therefore, the number of configurations remains unchanged while the number of configuration bits to be loaded is reduced, hence the number of test cycles. Table 6.6 summarizes the gain attained using proposed partial reconfiguration scheme for a single cluster test.

TABLE 6.6 RESULTS FOR 100% FAULT COVERAGE OF A CLUSTER

FPGA block	Using full reconfiguration		Using partial reconfiguration		Gain (%)
	#Config.	Test cycles	# Config.	Test cycles	
Crossbar 'Down'	60	3360	60	1440	57%
Crossbar 'Up'	24	1344	24	720	46%
CLB	4	224	4	128	42.85%

Another important advantage of partial reconfiguration technique is the reduction in memory size to store the test configurations. It is depicted in Figure 6.5 where a comparison is given in terms of memory requirement for CLB and cluster interconnect testing using full and partial reconfiguration schemes. The normalized results are obtained using actual memory size requirement for crossbar 'Up', crossbar 'Down' and CLB. As can be seen from the results, memory requirement is reduced by half when using partial reconfiguration for test schemes.

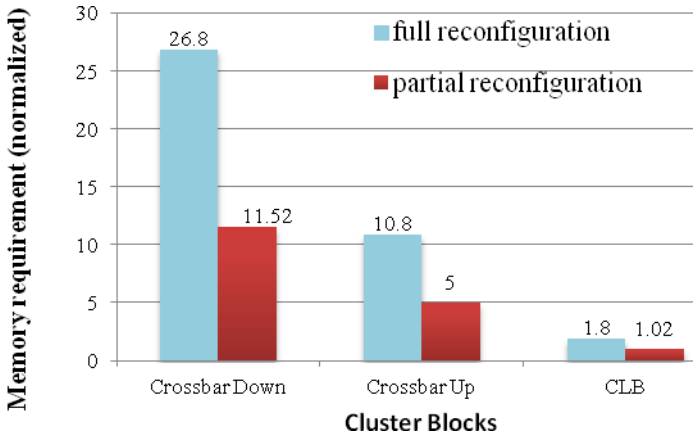


Figure 6.5: Memory requirement for full and partial test reconfigurations of a cluster

6.3 Impact of defect tolerant techniques on FPGA testability

In chapter 4, we discussed three defect tolerant schemes applied to FPGA under consideration. These techniques include *Butterfly* LUT design, Fine Grain Redundancy (FGR) and Distributed Feedback (DF) for cluster interconnect. The BIST techniques developed for the basic cluster are extended to the cluster with defect tolerant techniques.

Results are obtained in the form of number of test configurations required to obtain 100% fault coverage in the defect tolerant cluster.

The fault simulations are performed in the same way as it was done in the previous cases. Faults are injected on all potential nodes using gate-level netlist of the cluster. To emulate the BIST configuration, input constraints, test patterns are applied at the cluster primary inputs using automated script. For each configuration, the fault coverage is obtained. Table 6.7 summarizes of the BIST results for basic cluster, the cluster with *Butterfly* LUT, the cluster with FGR and the cluster enriched with DFs. It presents the number of test configurations required in each case along with the maximum achievable fault coverage.

TABLE 6.7 RESULTS FOR 100% FAULT COVERAGE OF A CLUSTER

Cluster architecture	# of test configurations	Max. achievable fault coverage
Basic	78	100%
With Fine Grain Redundancy	108	100%
With Distributed Feedback	88	100%
With *Butterfly*	82	85.6%

As compared to basic cluster architecture, cluster with FGR technique requires 38% and DF technique requires 12.8% increase of the test cost for the same maximum achievable fault coverage. The area overhead generated in the case of FGR and DF compared to basic cluster architecture is 33.3% and 20.4% respectively [Dhia 2013].

Similarly, results show that *Butterfly* design applied for LUTs increases the test cost up to 5% at the expense of 80% increase in LUT area while experiencing a drop of ~14% in testability.

6.3.1 Logical masking

In this section we will exploit the concept of logical masking to interpret the benefits of defect tolerant techniques employed earlier. Defect tolerance is referred to the design's inherent robustness against a defect. As a matter of fact, for some input combinations, failures caused by a defect within the cluster appear in a non-sensitized path and thus cannot be propagated to the output. This phenomenon is referred to as *logical masking*. As robustness metric for our work, we resort to logical masking.

Hypothesis: A defect is modeled by a stuck-at 0/1 at the output of a MUX2, and any MUX2 in the CLBs or crossbars can be defective.

Methodology: Defect injection is achieved through a platform that considers all possible input combinations and all possible locations of a single defect in a given design. As a matter of fact, the platform returns the number of logical masking. To get the logical masking rates, the numbers of logical masking is normalized by the total number of tests.

For quantitative analysis, we take benefit of the work done by [Dhia 2013] which presents the emulation results in terms of logical masking rates for all the cluster architectures explored in our work. Table 6.8 shows the logical masking rates per block for the basic cluster, with FGR and with butterfly architecture.

TABLE 6.8 LOGICAL MASKIGN EMULATION RESULTS [DHIA 2013]

Cluster architecture	**Logical masking per block (%)**		
	Crossbar 'Down'	**CLB**	**Crossbar 'Up'**
Basic	100	85.33	55.55
With *Butterfly*	100	100	55.55
With FGR	100	85.33	84.21
With DF	100	85.33	55.55
With Butterfly, FGR and DF	100	100	84.21

It is worth noting that the use of the *Butterfly* structure achieve complete tolerant to single defects in the case of the CLBs. And in the case of FGR in the interconnect blocks, logical masking rate was increased by roughly 30% (from 55.55 to 84.21 in case of crossbar 'Up'). Indeed, employing FGR in the crossbar 'Down' is seemingly pointless as long as the crossbar 'Down' is already 100% robust against single defects. Similar is the case with DF which doesn't improve masking compared to basic architecture. However, the benefits of FGR and DF are more pronounced when dealing with routability aspects which will be discussed in the next section. Combining all three hardening techniques enables to take advantage of the two gains in logical masking (i.e. in CLB and crossbar 'Up') but at the expense of area overhead [Dhia 2013].

6.3.2 Routability and defect avoidance

For a complete analysis of the defect tolerant technique used here, it is important to analyze their usefulness from a routability and defect bypassing view point. For this purpose, the work presented in [Amouri 2013] is helpful as it discusses the routability and the defect avoidance for the FGR and DF techniques on the similar cluster architecture.

We can define the routability of an FPGA architecture by the number of routing solutions it offers for an application to be mapped on it. The more interconnect resources in an FPGA architecture, higher is the routability. And higher the routability, more complex the applications that can be mapped. Moreover, it is also easier to bypass the defects. Defect bypassing (defect avoidance) consists of using spare resources instead of defective ones. Hence, defect avoidance engenders re-routing the application signals.

The methodology adopted by [Amouri 2013] is as follows. A defect, modelled by an *undefined* value at the Mux2 output in *Modelsim*®, is injected within the crossbar 'Down', which makes the cluster unusable. Then, the cluster is reconfigured to use either spare connection inside the defective crossbar; thanks to FGR or another crossbar 'Down'; thanks to DF. Hence, the defective Mux2 is bypassed and the cluster functionality is restored.

Since the inner architecture of the CLB has no impact on routability and defect avoidance, it is useless analyzing the *Butterfly* structure for its routability. Table 6.9 shows the maximum number of defective Mux2s that can be bypassed for each cluster design and the area overhead with respect to the basic cluster architecture.

TABLE 6.9 DEFECT AVOIDANCE AND ROUTABILITY RESULTS [AMOURI 2013]

Cluster Architecture	Total number of Mux2s	Number of bypassed Mux2s	Increase of cluster area(%)	Routable with one defect
Basic	588	0	0	No
With FGR	784	36	33.3	Yes
With DF	708	33	20.4	Yes
With FGR and DF	928	77	57.8	Yes

We can observe that DF and FGR allow bypassing virtually the same number of Mux2s (33 for the DF versus 36 for the FGR which represents a gain of only 0.5% in the overall cluster). As far as the architecture with DF is concerned, it is possible to increase

the number of distributed feedbacks but this would increase the cluster area by more than 20.4%. FGR also causes about 13% additional area overhead as compared to DF. Thus, if solely one hardening technique had to be used in the interconnect blocks, one would select the DF over the FGR. DF and FGR techniques can also be used together in the cluster which allows to bypass 77 Mux2s, that is more than the sum of bypassed Mux2s in the architectures using either FGR (36 bypassed Mux2s) or DF (33 bypassed Mux2s), but the cluster area is then increased by more than the sum of the overheads.

We can deduce from the above discussion that although the testability of the defect tolerant architecture is reduced or becomes costly compared to non-defect tolerant, if we take other phenomena into consideration such as logical masking, overall bypassing and routability, defect tolerant architecture could be a preferred choice.

6.4 Scan-DFT results

In order to evaluate the efficiency of the BIST architectures, a comparative analysis of the results discussed above with some "benchmarks" is necessary. For that purpose, we performed standard scan-design DFT simulations analyzing the effect of each hardening technique on the testability of the cluster. The testability metrics under consideration in this case include test time and the distribution of fault detected for a given test vector and the corresponding fault coverage.

In the following experiments, faults are injected at gate level netlist and the metrics of testability are measured for dominant faults, obtained after fault collapsing and equivalence. In each case, 40-50% of total faults are found to be redundant and thus removed from the fault list.

Figure 6.6 shows the distribution of the detected faults in the basic cluster architecture with respect to the number of required test patterns plotted along with the maximum achievable fault coverage. In the plot, fault coverage curve shots up at the beginning as the large number of faults are detected with a fewer number of test patterns, thanks to the deterministic algorithm of pattern generator. Later on, the ratio of the number of detected faults versus the number of required test patterns decreases which in turn gives further slower increase in fault coverage.

In other words, at the beginning of the test phase within a very short time, the test vectors applied are able to detect a high density of faults. The computational effort to detect more faults becomes higher and takes more processing time for relatively less number of faults detected and this lowers the slope of fault coverage.

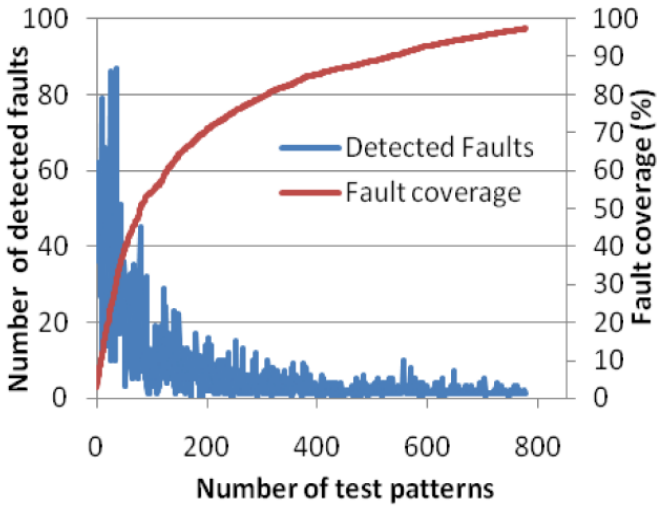


Figure 6.6: Distribution of detected faults and the corresponding fault coverage for the basic cluster vs. number of test patterns

Similarly, results for a cluster enriched with FGR technique are shown in Figure 6.7 As mentioned earlier, this architecture adds 33.3% area overhead and the potential faulty nodes are increased by exactly the same ratio. Considerably better fault coverage can be achieved as compared to the initial cluster at the cost of 12.5% extra test time.

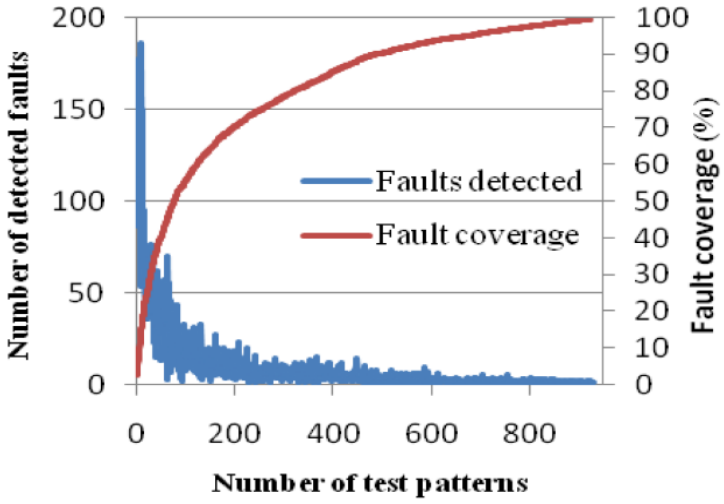


Figure 6.7: Distribution of detected faults and the corresponding fault coverage for the cluster with FGR vs. number of test patterns

Although feedback paths make fault detection costly in terms of computational time for test pattern generation as well as the number of required test patterns, DF technique gives a better trade-off in terms of testability as shown in Figure 6.8. For approximately

the same fault coverage, DF costs 8% less test time as compared to FGR. The main reason for this concession is the addition of potential nodes and devices in case of FGR which dramatically increases the number of faulty sites.

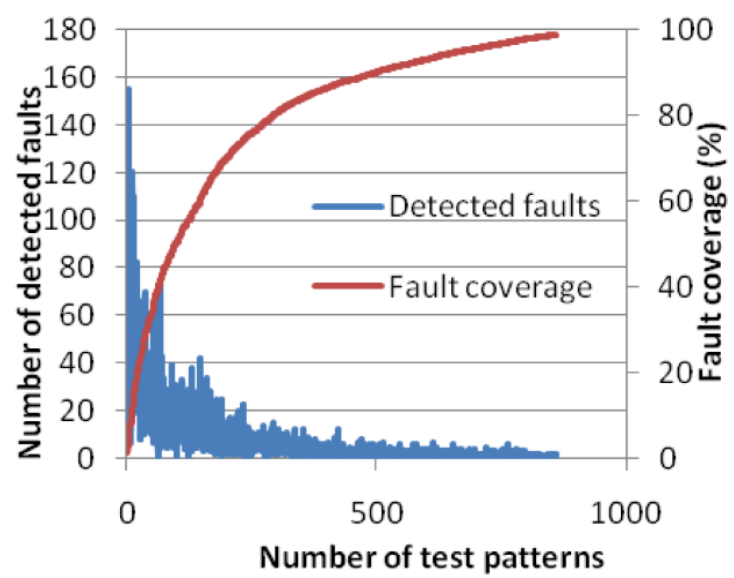


Figure 6.8: Distribution of detected faults and the corresponding fault coverage for the cluster with DF vs. number of test patterns

LUT-4 Butterfly design implemented in each CLB of the cluster degrades testability compared to the other hardening techniques. In spite of having less potential faulty nodes and relatively large number of test cycles, the maximum achievable fault coverage is about 84% (cf. Figure 6.9) which is significantly lower than all the other techniques. The reason is the requirement of high computational effort to generate effective test patterns such that faults present in the LUT structure can be propagated through the complex Butterfly structure, which in turn requires large number of test cycles for relatively lower number of injected faults as compared to DF or FGR.

Testability results for the cluster enriched with Butterfly are according to the expectations. Indeed, the observability of faults within Butterfly LUT-4 structure decreases because of the high rate of logical masking. Faults that are not detected are actually filtered out by the logical masking phenomena and do not result in an incorrect output. As a result, the design is more robust but less testable.

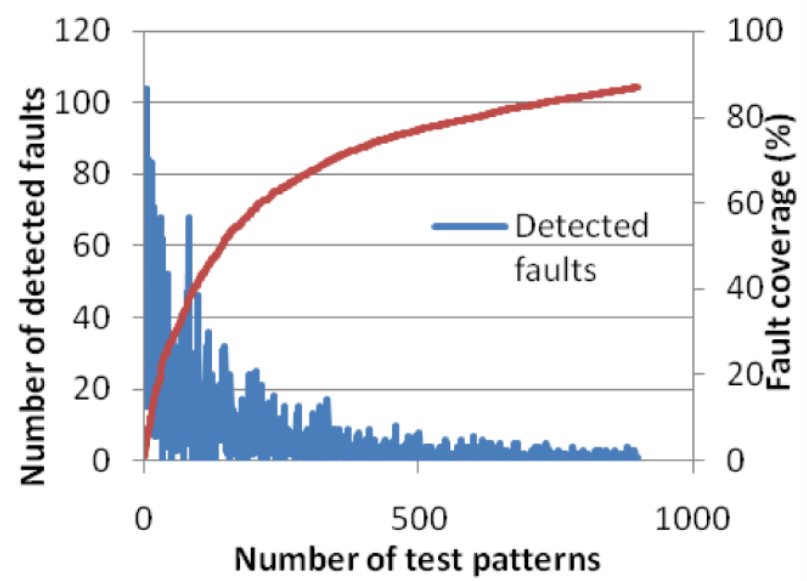


Figure 6.9: Distribution of detected faults and the corresponding fault coverage for the cluster with *Butterfly* LUT vs. number of test patterns

Interesting results are obtained for the architecture where all the above mentioned hardening schemes are combined in a cluster. Fault coverage drops drastically to 78% as shown in Figure 6.10. High ATPG computational effort is required to improve the fault coverage.

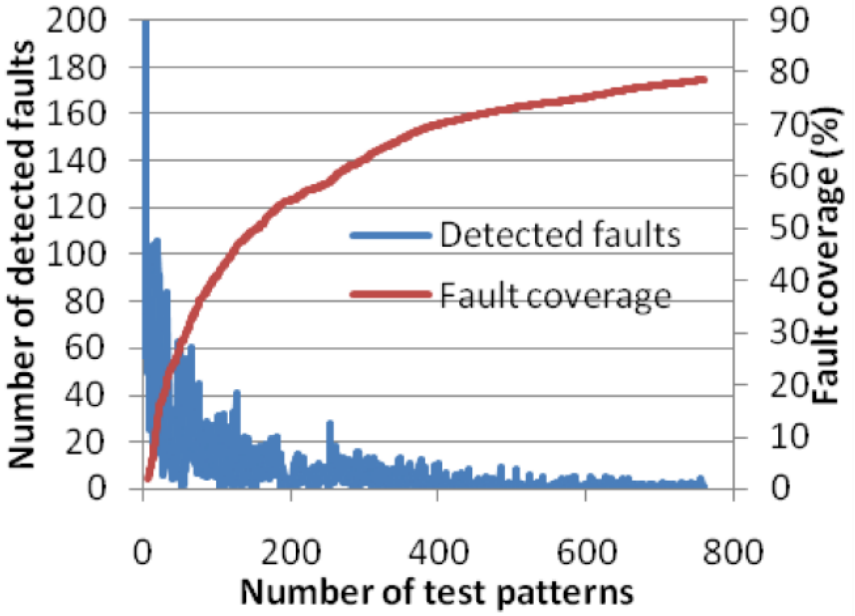


Figure 6.10: Distribution of detected faults and the corresponding fault coverage for the cluster with *Butterfly* LUT, FGR and DF vs. number of test patterns

Figure 6.11 depicts a comparison of the fault coverage and the respective test cost attained for different cluster architectures considered in this work. Both FGR and DF techniques give considerably high fault coverage. However FGR can be considered as the best solution if the test cost and robustness are taken into consideration. Table 6.10 shows the summary of the testability metrics for each hardening technique where the maximum achievable fault coverage and the respective number of required test patterns are given.

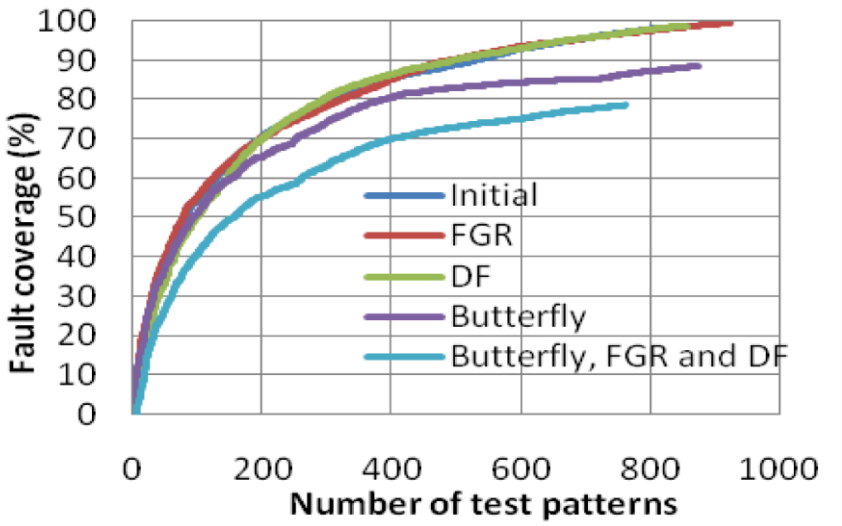


Figure 6.11: Comparison of fault coverage for the basic cluster with Butterfly and FGR.

TABLE 6.10 RESULTS FOR SCAN-DESIGN DFT

Cluster Architecture	Fault Coverage (%)	# Test patterns	# Test Cycles
Basic	97.99	808	10517
With FGR	99.52	923	12012
With DF	98.82	858	11167
With *Butterfly*	83.68	898	11697
With Butterfly, FGR and DF	78.22	761	9906

6.5 Comparison between FPGA BIST and scan-DFT

When using the scan-DFT results as "benchmark" for the validation of BIST simulation results, we can indeed appreciate the efficiency and scalability of the BIST

approach. These results are compared against scan-DFT results. BIST techniques implementation results are better in terms of test cost when the FGR and DF defect tolerant techniques are used and the testability degrades in case of *Butterfly* LUT design. The increase in test cost for FGR and DF using scan-DFT is 14% and 6.2% respectively compared to basic cluster architecture while in the case of BIST it represents only an increase of 4%. Similarly, the maximum achievable fault coverage in case of *Butterfly* LUT design is 83.68% using scan-DFT whereas BIST gives better results such as 85.6%. In short, scalability and better overall performance of FPGA BIST make it a better choice among available test methodologies.

6.6 Conclusion

In this chapter, a qualitative analysis of the proposed BIST schemes is described. The quality and efficiency of the test scheme is calculated in terms of maximum achievable fault coverage for a number of test configurations given by the BIST algorithm. Results are produced for various cluster sizes meanwhile analyzing the impact of cluster size on its testability. The test time optimization techniques introduced in this work are analyzed by extending the proposed BIST simulations which promises considerable gains in terms of test time. Moreover, BIST simulation results are obtained for the cluster enriched with defect tolerant techniques such as *Butterfly* implemented in the LUT design and Fine Grain Redundancy (FGR) and Distributed Feedback (DF) in the interconnect. A detailed comparison of defect tolerant cluster with the basic one, gives the increase of the test cost and the area overhead of the defect tolerant cluster. Considering the logical masking and defect bypass facilities, defect tolerant cluster is found to be a better choice. We also validated the BIST simulation results by using scan-DFT simulation results as "benchmarks" for each of the cluster architecture. Results show that the cluster of higher size i.e. 12, offers a better routability at a relatively less test cost along with the better robustness with FGR technique.

Chapter 7

Conclusion and perspectives

7.1 Conclusion

FPGAs are gaining a significant share in IC industry due to their reconfigurability and shorter time-to-market. High performance and low power features of the FPGAs make them a promising candidate for complex digital systems. However, this growing demand requires an increased reliability of the device. FPGA's reconfigurability on one hand is useful as it can be reprogrammed in a number of ways by the user for a given design. But on the other hand, it complicates the testing process for defects as the FPGA mapped device should be tested in all modes of operation to ensure a high reliability.

Many research teams performed studies on FPGA testing. Most of the works rely on Built-In Self-Test (BIST) approach for an exhaustive testing. Several methods have also been proposed to minimize the test cost which is essentially the test time in case of FPGAs. Some of the test time reduction methods require major modification in the FPGA's original architecture to make them self reconfigurable. In addition to this, most of the existing works on FPGA testing propose test methodologies for commercial FPGAs. FPGA's logic and interconnect architecture optimization is an ongoing process which requires thorough analyses and solutions to the problems associated with it such as testability, routability and robustness of the developing FPGA.

This thesis presents test and diagnosis schemes for a novel interconnect topology and logic blocks in a mesh SRAM-based FPGA. The proposed BIST schemes provide a generic and scalable solution to implement it on any size of the FPGA array. This manuscript provides an overview of the work done in this regard.

- The first contribution of this thesis involves the development of test and diagnosis algorithms to detect and locate faults in the logic and interconnect resources of the novel FPGA. This FPGA is mainly composed of clusters and switch boxes arranged in a grid. In clusters, 'n' number of identical logic blocks are grouped together. The switch boxes are composed of multiplexer-based crossbars forming

global interconnect. The interconnect structures present in the cluster and the switch box are made of sparsely populated crossbars. The prominent feature of the switch box is its hierarchical interconnect topology based on unidirectional network. We adopted BIST approach to define the test configurations for each block in the cluster as well as in the switch box. Our aim was to perform an exhaustive testing of every MUX in the logic block and interconnect with the least possible configurations. For that purpose, specific BIST structures were developed. These structures are composed of clusters that are configured as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) to test a Block Under Test (BUT) during a test phase. In each test phase, a number of configurations were applied to test a specific part or path of the BUT which formed a test session. At the completion of a test session, TPG/ORA may require to be reconfigured forming the next session and it continues until the BUT is completely tested. Similarly, a number of test phases are required to fully test the cluster and switch box of the FPGA. The BIST strategy we proposed for the clusters involves only two test sessions. In each test session, half of the FPGA is configured as TPG/ORA while half as BUT. Moreover, we utilized two TPGs and multiple ORAs mechanism to avoid fault masking in case of faulty TPG or comparing BUT pair. However, in the proposed structure, one fourth of the BUTs are analyzed in single ORAs while rest are analyzed twice in two different ORAs. This may cause fault masking problem for BUTs that are compared once and thus is a limitation of the proposed BIST scheme for the cluster.

- The proposed strategy for testing multilevel interconnect at switch box is based on the selection of a number of Paths Under Test (PUTs) in a test phase using only adjacent logic resources. Using this strategy, any NxN FPGA array can be further tested by N parallel 2x2 array procedure which ultimately reduces the test time. The proposed BIST structures for switch box are not subjected to any limitation as far as fault masking is concerned. PUTs are compared and analyzed twice in order to ensure the fault detection and correct diagnosis. For fault diagnosis, a method was developed requiring the manipulation of the ORA sequence and fault mapping between the current configuration and the corresponding fault list produced by the tools developed for this purpose.

- The fault models considered in this work include single stuck-ats and pair-wise bridging faults. The BIST structures and configurations were developed targeting these fault models. Both cluster and switch box BIST schemes may detect some

multiple or clustered faults but in order to fully test and diagnose such types of faults, more test configurations will be required. Similarly, ORA functions may change and the configuration of TPGs may also be different as the specific types of test patterns will be needed to detect clustered or multiple faults.

- To address the FPGA architectural optimization with respect to routability, the proposed BIST schemes were extended to determine the impact of size of the cluster on its testability. For that purpose, cluster of size 4, 6, 8, 10 and 12 were analyzed. Although, the number of test configurations increases with increasing the cluster size as expected but the increment in number of configurations relative to increase in cluster size was not linear. For example, increasing cluster size from 4 to 6 and from 6 to 8 costs 36% and 13% more test configurations respectively to achieve 100% fault coverage. Similarly, increasing cluster size from 8 to 10 costs 61% more test configurations. It is interesting to mention that cluster size increment from 10 to 12 costs just 11%. This is due to the fact that cluster size 6 and 10 has more redundant interconnects to retain the architectural symmetry. Therefore, from testability point of view, cluster of size 8 and 12 were found to be test efficient as compared to cluster size 6 and 10. Moreover, it also shows that the BIST schemes are equally applicable to the larger cluster size i.e. 12 and give better results.

- Two strategies for test time optimization were proposed. The first one involves joint testing of logic blocks and intra-cluster interconnect. In this scheme, the number of configurations required for CLB testing was saved as CLBs were tested during testing of crossbar 'Up' of the cluster. For that purpose, a combination of CLB's configurations was worked out which is different than the CLBs' configurations during separate testing. Results show that significant reduction in the number of test configurations can be achieved for smaller cluster sizes. Such as, the number of test configurations for cluster of size 4 and 6 reduces by 14% and 9.3% respectively. We note that as the cluster size increases, reduction in number of test configurations decreases. It is due to the fact that the number of configurations saved becomes smaller compared to the total number of test configurations as cluster size increases. Therefore, cluster of size 12 saves merely 4.5% of total test configurations by using this optimization strategy.

- The other strategy proposed for test time optimization uses partial reconfiguration mechanism in which only the configuration bits needed to be modified are loaded

into the FPGA, keeping the other configuration unchanged. This scheme does not reduce the total number of test configuration required for exhaustive testing but it reduces the number of bits to be loaded into the FPGA during a test configuration which surely reduces the download time. Moreover, it also reduces the memory requirement for storing configuration bits. An important feature of the proposed BIST algorithm is that it does not have any repetition of the test configurations. Thanks to the BIST validation flow which helps in minimizing the number of redundant configurations. Results produced for cluster partial reconfiguration shows the gain of 54% in overall test time as compared to full reconfiguration.

- Defects in FPGAs can be avoided by reconfiguring the application on the fault-free resources. It can only be done if fault location is known as well as the extra fault free resources are available. Logic and interconnect redundancy techniques are usually employed to make FPGA defect tolerant. The defect tolerant techniques considered in this thesis include *Butterfly* LUT design as well as Fine Grain Redundancy (FGR) and Distributed Feedback (DF) at the cluster interconnect. Chapter 4 gives a brief overview on the enrichment of basic FPGA architecture with these defect tolerant techniques. The proposed BIST schemes were extended and analyzed for the increase in test cost and the impact on overall testability in the defect tolerant FPGAs. We found that *Butterfly* LUT design masks single stuck-at faults giving a considerable decrease in fault coverage (max. achievable fault coverage 85.6%) even with 5% increase in the number of test configurations as compared to basic LUT design. However, BIST schemes were able to achieve 100% fault coverage for both FGR and DF techniques. FGR and DF increases the test cost as expected by 38% and 12% respectively compared to basic cluster design. When taking logical masking and better routability into account, defect tolerant FPGA is better from testability viewpoint than the basic FPGA without any defect tolerance.

- Another major contribution of this thesis is the development of tools to integrate the BIST schemes into the standard FPGA CAD flow. As a matter of fact, the commercially available FPGA tools do not support any FPGA architecture other than their own. Therefore, implementing the proposed BIST schemes using non-commercial tools for a given FPGA comes with challenges. For this purpose, a set of tools were developed which translate the BIST algorithms into files acceptable by a non-commercial FPGA tool and integrate them into VTR-FPGA flow. VTR (Verilog-To-Routing) Project is a reliable open source platform developed by

multiple university research groups for FPGA research purposes. To perform fault mapping and qualitative analysis, another set of tools were developed integrating them into so called validation flow using Synopsys TetraMAX®. The developed tools are generic and allow users to provide the architectural parameters of the block under test. However, developed tools are limited to the FPGA architecture considered in this work.

- Finally the chapter 6 presents the results obtained by simulation of the BIST schemes for various FPGA architectures. It involved the FPGA with different cluster sizes as well as the FPGAs enriched with defect tolerant techniques. Moreover, the cost of defect tolerant FPGA in terms of routability and defect avoidance was also taken into account to fully analyze the overall testability. Similar results were obtained by performing scan-based DFT simulations to have a comparative analysis of the proposed BIST and standard scan-based DFT technique. Using scan-based DFT, maximum fault coverage achieved for the basic cluster design is 97.99% while it is 100% in case of BIST. Although, fault coverage improved to 99.52% for FGR but the increase in test cost by FGR using scan-based DFT is 14% while BIST just increases it by 4%. In case of DF, fault coverage achieved using scan-based is 98.82% along with the increase in test cost by 6.1%. Similarly, maximum achievable fault coverage in case of *Butterfly* LUT design is 83.68% using scan-based whereas BIST gives better as 85.6%. It is due to the fact that ATPG uses deterministic algorithm for pattern generation while BIST uses pseudo-exhaustive. It requires more computational effort for ATPG in scan-based DFT to increase fault coverage.

7.2 Perspectives

This thesis provides the basis for the development of test and diagnosis schemes for hierarchical mesh of cluster FPGA. As the proposed schemes are scalable, they can be applied to any size of the FPGA having similar architecture without any modification. There are many aspects of the FPGA testability that remained untouched in this work and can be considered for the future work.

The defect tolerant schemes presented here can be implemented on switch boxes too in addition to the clusters and the proposed test methods can be extended to test the defect tolerant switch boxes as well. The main block in switch box is a crossbar which is to be enriched with defect tolerant schemes. Thus, the proposed BIST schemes for defect tolerant crossbar in a cluster can be used for switch box too.

There are several other defect tolerant techniques such as Adapted Fine Grain Redundancy (AFGR), coarse grain redundancy (e.g. direction connection between clusters) which can be used to improve FPGA robustness and therefore requires a thorough analysis of their impact on the FPGA testability. For that purpose, the proposed BIST schemes can be extended and modified to apply to the new defect tolerant blocks in the FPGA.

There is always a requirement to improve the existing test time reduction methods and to develop new optimization techniques. Faster and less expensive test methods are always desired. Our work can provide the basic reference for any other optimization technique embedded in this FPGA architecture to improve test time.

The BIST strategies and tools developed during this work provide a useful platform to extend the testing to other blocks of the FPGA such as memory-cells, I/O pads etc. The proposed BIST schemes can be enhanced and improved to perform online test and delay test. Future work should also focus on testing FPGAs for Single Event Upsets (SEUs) which has been an interesting topic for the last few years in the FPGA research field.

Appendix A

Mesh of clusters FPGA architecture perspectives

This appendix briefly describes the prominent features of the mesh of clusters FPGA considered in this thesis. The hierarchical topology in the mesh of cluster FPGA is inspired by the classical hierarchical FPGA. In the following, the architecture of the hierarchical FPGA is first presented. Then the advantages of the mesh of cluster over classical mesh architecture are discussed.

The hierarchical FPGA

Modern FPGAs use clustering of logic blocks to improve routing area and signal propagation through the routing network. Since most logic designs exhibit locality of connections, which implies a hierarchy in placement and routing of connections between logic blocks, gathering the latter into clusters provide smaller routing delays. The number of switches can also be reduced by depopulating the interconnect structure, e.g. by using sparse rather than full crossbars into the cluster. Connection blocks, an intermediate level of multiplexers connecting the routing tracks to input multiplexers of logic blocks, which are used in the VPR architecture [Betz 1999] and the Altera Stratix architecture [Lewis 2005], can be avoided by connecting the routing tracks directly to the input multiplexers, as in the Xilinx Virtex architectures [Xilinx]. Another possibility to avoid the use of connection blocks is to connect clusters directly to switch boxes [Marrakchi 2010].

Figure A.1 depicts a hierarchical FPGA topology. Several commercial FPGAs have a hierarchical topology, such as Altera Apex [Hutton 2001] with two levels of hierarchy. The concept of hierarchical FPGA architectures has recently interested academic researchers. Many hierarchical architectures have been proposed in [Wang 1998] [Aggarwal 1994] [Marrakchi 2008]. In [Marrakchi 2008], an efficient tree topology for the FPGA interconnect network is introduced and demonstrated a gain of 40% in total area compared to a mesh topology.

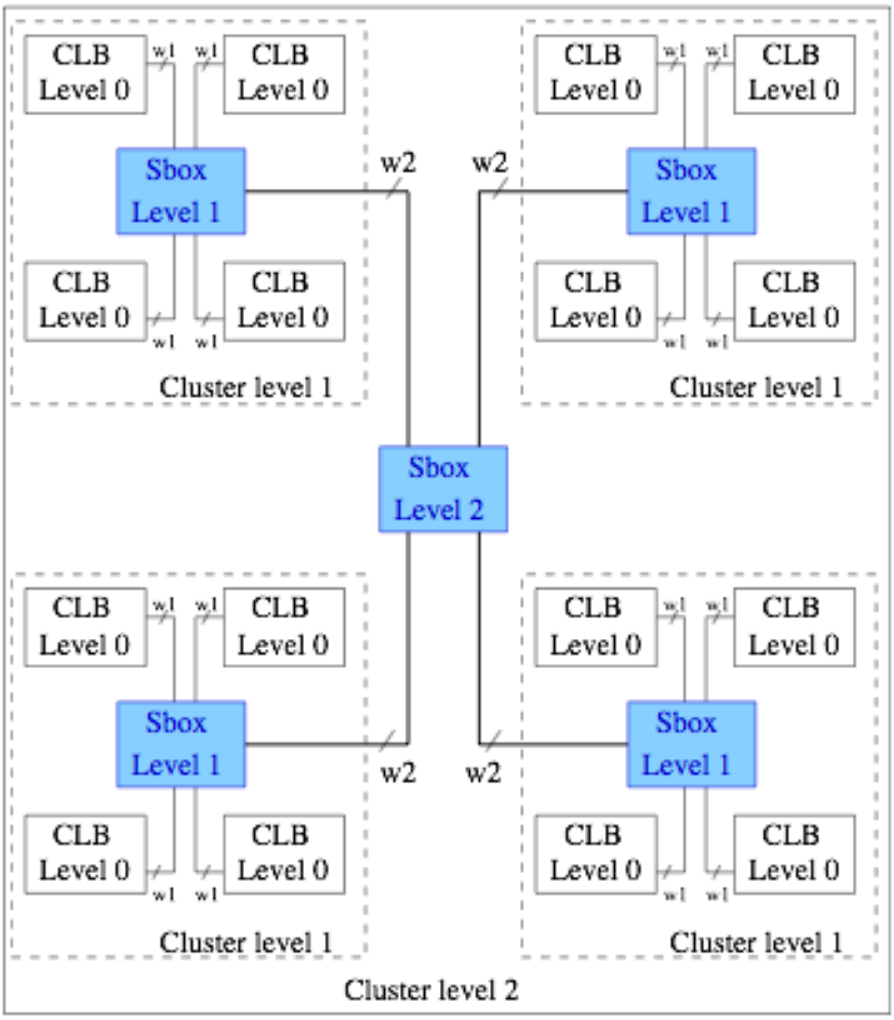


Figure A.1: Hierarchical FPGA architecture

Efficiency of mesh of clusters FPGA: Area and routability

Here, a comparative analysis of the area efficiency and routability of mesh of cluster and classical VPR FPGA architecture is presented. The architectures of both the FPGAs are explained in chapter 3 of the thesis. Here, the cluster size of 8 and LUT size of 4 is considered for both architectures. The VPR architecture uses a unidirectional routing network with single length segments and a Wilton switch block. Each cluster logic block contains I inputs and 8 outputs which are distributed over the cluster sides. LUTs pins are connected to cluster pins using a full local crossbar. Connection block population is defined by Fc_{in} and Fcout parameters, where Fc_{in} is routing channel to cluster input switch density and Fc_{out} is cluster output to the routing channel density. For both architectures, the smallest architecture implementing every MCNC benchmark circuit is determined.

Table A.1 shows the architecture size and percentage gain in the number of switches and the area when implementing benchmark circuits required to implement each of the benchmark circuits over the VPR and the new mesh of clusters FPGAs.

Table A.1: Area and switch utilization gain in VPR clustered and mesh of clusters FPGA

MCNC	VPR Mesh Cluster Size 8		New Clustered Mesh Cluster Size 8		Gain	
	SW x10^3	Area (λ^2) x10^6	SW x10^3	Area (λ^2) x10^6	SW %	Area %
alu4	390	1246	216	613	44	50
apex2	444	1278	292	828	34	35
apex4	281	812	204	557	27	31
bigkey	332	979	171	486	48	50
clma	2328	6576	1386	3964	40	39
des	325	953	196	556	39	41
diffeq	277	815	183	520	33	36
dsip	349	1023	158	449	54	56
elliptic	915	2606	489	1393	46	46
ex5p	1226	3477	723	2060	41	40
ex1010	235	683	178	505	24	26
frisc	912	2600	505	1436	44	44
misex3	326	943	190	541	41	42
pdc	1491	4166	862	2468	42	40
s298	435	1256	226	642	48	48
s38417	1371	3978	749	2139	45	46
s38584	1312	3830	783	2148	40	43
seq	392	1277	261	742	33	41
spla	1085	3049	633	1803	41	40
tseng	200	588	111	313	44	46
Average	**726**	**2106**	**425**	**1208**	**41**	**42**

As the FPGA area and routability depend on the Channel Width (CW), multiple CW values are exercised to determine the optimal architecture. an automated router is used to find the minimum CW for each bench according to the dichotomy algorithm. For example, the router starts routing with CW= 32, if the bench is routable, CW is reduced to 16 for the same bench. Then, if it becomes non-routable with CW = 16, the router tries with 24 and so on. Therefore, the final value represents the minimal CW to route the bench in a given FPGA. The average of the FPGA area and the channel width used for implementing MCNC benchmark circuits is given at the bottom of Table A.1.

Interesting results are obtained when observing the number of switches used in both case. In the case of "tseng" the smallest circuit, 44% switches reduction and 40% in the case of "clma" the largest circuit is achieved. Thus the new cluster-based interconnect is attractive for both small and large circuits. An average of 41% reduction of the switches number is achieved for new mesh of cluster FPGA.

To find the architecture that can implement all benchmarks, all circuits are placed in the biggest array for both architectures (33x33 for the VPR Mesh and 36x36 for the new mesh architecture) and the largest of the minimum channel width used is determined. Table A.2 shows the channel width for 20 MCNC benchmarks placed and routed in the biggest array. It is concluded that the maximum channel width used is equal to 84 and 46 in the VPR clustered Mesh and in the mesh of cluster architecture respectively. So, the new mesh architecture can implement all circuits with a gain of 45% in area compared to VPR Mesh.

To maintain the consistency of the analysis given in the thesis, the impact of cluster size on the area of the new mesh of cluster FPGA is presented. Table A.3 (a&b) shows the minimal architecture size (Nx X Ny), the minimal FPGA area and the minimal CW to route each bench.

It can be noticed that the architecture size decreases when the cluster size increases since with more CLBs per cluster, fewer clusters in the FPGA are needed. In Table II, Opt represents the minimal architecture in terms of array size, area and CW which allows to route all benchmark circuits. With cluster of size 8, 80% of the benchmark circuits can be implemented on a smaller architecture as compared to the cluster of size 4, 6, 10 and 12. But if we compare the optimal architecture, the largest bench (clma) is routed on a bigger architecture in the case of cluster size 4, 6, 8 and 10 as compared to the cluster size 12.

Table A.2: Max. channel width and area utilization in VPR clustered and mesh of clusters FPGA for implementing MCNC benchmarks

MCNC benchmarks	VPR Clustered Mesh 33x33	VPR Clustered Mesh 36x36
alu4	50	28
apex2	50	30
apex4	46	30
bigkey	34	22
clma	72	46
des	32	18
diffeq	32	24
dsip	38	20
elliptic	58	32
ex5p	62	36
ex1010	44	32
frisc	58	34
misex3	46	30
pdc	84	44
s298	46	26
s38417	44	28
s38584	40	28
seq	50	32
spla	78	38
tseng	28	22
Max. W	**84**	**46**
Max. Switches No.	**2608**	**1382**
Max. Area	**7281**	**3954**

Furthermore, in terms of routability, an architecture of cluster size 12 can route all benches with the smallest architecture (29 x 29). That is why, for optimal (Opt) results, the minimal area able to route all benches is $3883 \times 10^6 \lambda^2$ which corresponds to the cluster of size 12.

The above discussion gives a brief overview of the area efficiency of the interconnect topologies in mesh of cluster and VPR mesh FPGA. It signifies the advantages of the hierarchical mesh of cluster FPGA both in terms of routability and the area compared to classical VPR FPGA.

Table A.3: Max. Channel width and area utilization for implementing MCNC benchmarks in mesh of clusters FPGA with different cluster sizes.

Table A.3 (a): For cluster size 4, 6 and 8.

MCNC Bench-Marks	Cluster Size 4			Cluster Size 6			Cluster Size 8		
	N_x *Ny	Area(.10^6 λ^2)	CW min	N_x *Ny	Area (.10^6 λ^2)	CW min	N_x *Ny	Area (.10^6 λ^2)	CW min
alu4	21*21	615.5	24	20*20	691.5	22	16*16	613.5	30
apex2	24*24	852.9	26	23*23	960.4	24	18*18	828.3	34
apex4	20*20	560.6	24	19*19	657.4	24	15*15	557.5	34
bigkey	22*22	638	14	20*20	608.2	14	15*15	486	24
clma	49*49	4711	30	46*46	4698	34	36*36	3964	46
des	22*22	629.3	16	20*20	692.9	16	16*16	556.2	24
diffeq	21*21	502	18	19*19	528.7	16	15*15	520.2	28
dsip	20*20	554.9	16	18*18	517.1	16	15*15	449.5	20
elliptic	32*32	1509	26	29*29	1522	24	23*23	1393	36
ex5p	19*19	506.3	24	18*18	591.8	24	14*14	505.3	34
ex1010	38*38	2007	24	36*36	2341	24	28*28	2060	36
frisc	32*32	1595	28	30*30	1704	26	23*23	1436	38
misex3	21*21	615.5	24	19*19	657.4	24	15*15	541.7	30
pdc	38*38	2719	36	37*37	3038	34	28*28	2468	48
s298	23*23	602.1	18	21*21	723	20	17*17	642.3	26
s38417	41*41	2332	24	37*37	2353	22	30*30	2139	30
s38584	41*41	2258	20	35*35	2006	20	29*29	2148	34
seq	23*23	739.9	24	22*22	879.3	24	17*17	742.1	34
spla	34*34	1989	32	33*33	2328	32	25*25	1803	42
tseng	17*17	327	16	15*15	335.6	16	12*12	313.3	24
Opt	**49*49**	**4711**	**30**	**46*46**	**4698**	**34**	**36*36**	**3964**	**46**

Table A.3 (b): For cluster size 10 and 12.

MCNC Bench-Marks	Cluster Size 10			Cluster Size 12		
	$N_x * N_y$	**Area ($.10^6 \lambda^2$)**	**CW min**	**$N_x * N_y$**	**Area ($.10^6 \lambda^2$)**	**CW min**
alu4	15*15	707.6	32	13*13	656.7	40
apex2	18*18	987.7	30	15*15	911.3	44
apex4	15*15	707.6	32	13*13	673.5	42
bigkey	14*14	530.1	20	13*13	546.9	24
clma	34*34	4302	46	29*29	3883	58
des	15*15	623.6	22	13*13	591.9	30
diffeq	14*14	532.6	22	12*12	497.5	30
dsip	14*14	530.1	20	11*11	403.9	26
elliptic	22*22	1558	34	19*19	1487	46
ex5p	14*14	616.4	32	12*12	561.9	40
ex1010	28*28	2516	34	23*23	2114	44
frisc	23*23	1701	34	19*19	1487	46
misex3	15*15	689.3	30	13*13	656.7	40
pdc	28*28	2915	46	24*24	2707	60
s298	16*16	734.7	26	14*14	691.9	32
s38417	28*28	2235	26	24*24	2120	36
s38584	26*26	2062	30	24*24	2281	42
seq	17*17	907	32	14*14	794.7	44
spla	25*25	2171	40	21*21	1887	50
tseng	12*12	396.7	22	10*10	360.4	32
Opt	**34*34**	**4302**	**46**	**29*29**	**3883**	**58**

Appendix B

BIST integration tools

BIST implementation starts with the RTL description of the BIST modules (i.e. TPG/ORA and BUT), defining the circuits required to produce the desired type of test patterns and type of response analyzer. This description is then used to provide information about logic resources needed to implement the desired function. Specific placement constraints are given to place the BIST modules (TPG/BUT/ORA) at desired positions in the FPGA. Similarly, for each configuration, routing constraints *(.desc files*) are given to route the signal in targeted paths. Every configuration corresponds to a unique bitstream. For a test scheme scalable to any FPGA array size, automation of test procedures is a big requirement. Therefore, automated tools are developed to generate the constraint files which are then used to produce the required BIST bitstream for user given architectural parameters (e.g. cluster size, FPGA array size, etc).

A set of tools is based on Perl and C++ source codes is developed which produce the output files required to perform the test and diagnosis of the FPGA. As mentioned earlier, BIST is a modular based technique where different blocks are tested in different configurations. Therefore, to simplify the implementation procedures, separate constraints files and scripts are produced for different FPGA blocks. Moreover, all codes are generic and deals with cluster/FPGA size, cluster index, cluster I/Os etc. Figure B.1 shows the integration of the constraint files produced by the tools in the BIST implementation and verification flows.

Constraint files (.desc)

For BIST implementation, constraints files are given to the Placer and Router in the VPR flow. The constraint files are descriptive files that define the placement of BIST modules and the connection among them. The input required to produce such files include, FPGA array size and cluster size, name and size of the module to be tested.

Script files (.tcl)

These files are used in the verification flow. These file are composed of the commands to run the fault simulation tool. These files include the input constraints which emulate the BIST configuration for under test block of the FPGA. For each BIST configuration, a unique .tcl file is required.

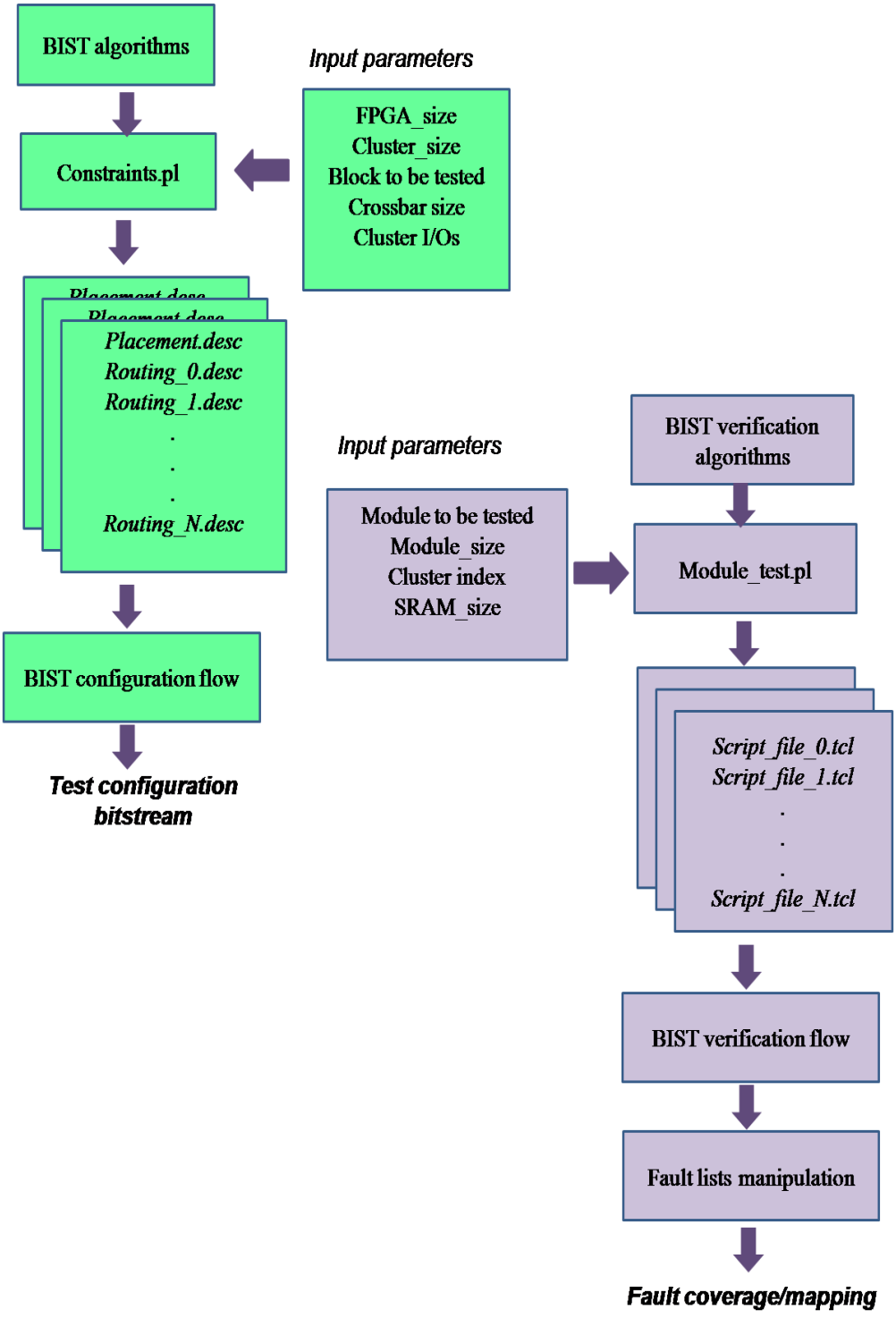


Figure B.1: BIST integration in BIST implementation and verification flows.

Glossary

BIST	Built-In Self-Test
BUT	Block Under Test
CAD	Computer Aided Design
CLB	Configurable Logic Block
CUT	Circuit Under Test
DF	Distributed Feedback
DMSB	Down Mini Switch Box
FGR	Fine Grain Redundancy
FSM	Finite State Machine
LFSR	Liner Feedback Shift Register
LUT-4	4-input Look Up Table
MUX	Multiplexer
ORA	Output Response Analyzer
SA0	Stuck-at 0
SA1	Stuck-at 1
STAR	Self Test ARea
TMR	Triple Modular Redundancy
TPG	Test Pattern Generator
UMSB	Up Mini Switch Box
VPR	Versatile Packing and Routing
VTR	Verilog-To-Routing
WUT	Wire Under Test

List of Publications

- **Saif Ur Rehman**, Mounir Benabdenbi and Lorena Anghel, " Application-Independent Testing of Multilevel Interconnect in Mesh-Based FPGAs," International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS), pages: 1-6, April 2015.

- **Saif Ur Rehman**, Adrien Blanchardon, Arwa Ben Dhia, Mounir Benabdenbi, Roselyne Chotin-Avot, Lirida Alves de Barros Naviner, Lorena Anghel, Habib Mehrez, Emna Amouri and Zied Marrakchi, "Impact of Cluster Size on Routability, Testability and Robustness of a Cluster in a Mesh FPGA," IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pages: 553-558, July 2014.

- **Saif Ur Rehman**, Mounir Benabdenbi and Lorena Anghel, "Cost-efficient testing of a cluster in a mesh SRAM-based FPGA, " International On-Line Testing Symposium (IOLTS), pages: 75-80, July 2014.

- **Saif Ur Rehman**, Mounir Benabdenbi and Lorena Anghel, "Test and diagnosis of FPGA cluster using partial reconfiguration," Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pages: 1-4, June 30 2014 - July 3 2014.

- Arwa Ben Dhia, **Saif Ur Rehman**, Adrien Blanchardon, Lirida Alves de Barros Naviner, Mounir Benabdenbi, Roselyne Chotin-Avot, Habib Mehrez, Emna Amouri and Zied Marrakchi, "A Defect-tolerant Cluster in a Mesh SRAM-based FPGA," Proceedings of International Conference on Field Programmable Technology (FPT), pages: 434-437, December 2013.

- **Saif Ur Rehman**, Mounir Benabdenbi and Lorena Anghel, "BIST for logic and local interconnect resources in a novel mesh of cluster FPGA," International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), pages: 296-301, October 2013.

References

[Abramovici 1999] M. Abramovici, C. Stroud, S. Wijesuriya, C. Hamilton, and V.V., 1999. Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications. In *Proc. International Test Conf.* pp. 973–982.

[Abramovici 2001] Abramovici, M. & Charles, E., 2001. BIST-Based Test and Diagnosis of FPGA Blocks. *IEEE Trans. on VLSI Systems*, 9(1), pp.159–172.

[Abramovici 2003] Abramovici, M. & Stroud, C.E., 2003. BIST-based delay-fault testing in FPGAs. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 19(5), pp.549–558.

[AFSoC] Antifuse FPGAs. *http://www.microsemi.com/products/fpga-soc/antifuse-fpgas*.

[Aggarwal 1994] Aggarwal, A.A.; Lewis, D.M. , Routing architectures for hierarchical field programmable gate arrays, Computer Design: VLSI in Computers and Processors, ICCD '94. Proceedings., IEEE International Conference on, pp 475-478, 1994

[Ahmed 2004] Ahmed, E. & Rose, J., 2004. The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(3), pp.288–298.

[Almurib 2014] Almurib, H.A.F., Kumar, T.N., Member, S. & Lombardi, F., 2014. Scalable Application-Dependent Diagnosis of Interconnects of SRAM-Based FPGAs. , 63(6), pp.1540–1550.

[Altera] QUARTUS Altera. *https://www.altera.com/products/design-software/fpga-design/quartus-ii/overview.tablet.html*.

[Amouri 2013] Amouri, E., Blanchardon, A., Chotin-avot, R., Mehrez, H., et al., 2013. Efficient Multilevel Interconnect Topology for Cluster-based Mesh FPGA Architecture. International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp 1-6, 2013

[Atmel] FPGA Integrated Development Systems (IDS). *http://www.atmel.com/tools/fpgaintegrateddevelopmentsystems_ids_.aspx?tab=devices*.

[Ban 2010] Tian Ban; de Barros Naviner, L.A. , A simple fault-tolerant digital voter circuit in TMR nanoarchitectures, NEWCAS Conference (NEWCAS), 2010 8th IEEE International, pp 269 - 272, 2010

[Betz 1999] Betz, V. & Rose, J., 1999. FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density. *International symposium on Field programmable gate arrays*, pp.59–68. Available at: http://dl.acm.org/citation.cfm?id=296428.

[Bushnell 2000] Bushnell, M.L. & Agrawal, V.D., 2000. *Essentials of Electronic Testing*, Kluwer Academic Pulishers.

[Dhia 2012] Ben Dhia, A.; Naviner, L.; Matherat, P. A new fault-tolerant architecture for CLBs in SRAM-based FPGAs, *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on, pp 761-764, 2012*

[Dhia 2013] A. B. Dhia, S. Pagliarini, L. de B. Naviner, H. Mehrez, and P. Matherat, "A defect-tolerant area-efficient multiplexer for basic blocks in SRAMbased FPGAs ," Microelectronics Reliability, vol. 53, no. 9-11, pp. 1189 – 1193, 2013.

[Dhingra 2005] S. Dhingra, S. Garimella, A. Newalker, and C.S., 2005. Built-In Self-Test of Virtex and Spartan-II FPGAs using partial reconfiguration. In *Proc. North Atlantic Workshop*. pp. 7–14.

[Doumar 1999] Doumar, A.; Ito, H. , Design of an automatic testing for FPGAs, European Test Workshop 1999. Proceedings, pp 152-157.

[Doumar 2000] A. Doumar, T. Ohmameuda, and H.I., 2000. "Fast testable design for SRAMbased FPGAs,. *IEICE Trans. Inform. Sys.*,, pp.116–127.

[Dutt 2008] Dutt, S., Verma, V. & Suthar, V., 2008. Built-in-Self-Test of FPGAs With Provable Diagnosabilities and High Diagnostic Coverage With Application to Online Testing. , Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 27(2), pp.309–326.

[Dutton 2009] Dutton, B.F. & Stroud, C.E., 2009. Built-In Self-Test of Configurable Logic Blocks in. In *Proc. IEEE Southeastern Symp. on System Theory*. pp. 230– 234.

[Dutton 2009b] Dutton, B.F. & Stroud, C.E., 2009. Built-in self-test of configurable logic blocks in virtex-5 FPGAs. *2009 IEEE International Symposium on Sustainable Systems and Technology, ISSST 2009*, pp.230–234.

[Farooq 2008] Farooq, U., Marrakchi, Z., Mrabet, H., Mehrez, H., et al., 2008. The Effect of LUT and Cluster size on a Tree based FPGA Architecture. *international conference on reconfigurable computing and FPGAs*, pp.115–120.

[Feng 2007] W.Feng and S.Kaptanoglu., 2007. Designing efficient input interconnect blocks for LUT clusters using counting and entropy. In *International Symposium on Field Programmable Gate Array*. pp. 23–32.

[FlexRAS] https://www.mentor.com/products/fv/flexras

[Fernandes 2003] D. Fernandes and I. Harris, 2003. Application of built-in self test for interconnect testing of FPGAs. In *Proc. Int. Test Conf.* pp. 1248–1257.

[Guccione 2001] Guccione, S.A. & Levi, D., 2001. *JBits : A Java-Based Interface to FPGA Hardware*,

[Gusmão 2004] Gusmão, F., Kastensmidt, D.L., Neuberger, G., Hentschke, R.F., et al., 2004. Designing Fault-Tolerant Techniques for SRAM-Based FPGAs. Design & Test of Computers, IEEE , pp.552–562.

[Harris 2000] Harris, I. and Tessier, R., 2000. Diagnosis of interconnect faults in cluster-based FPGA architectures. In *IEEE/ACM Int. Conf. Computer-Aided Design*. pp. 472–475.

[Harris 2001] Harris, I., Menon, P., and Tessier, R., "BIST-based delay path testing in FPGA architectures," *Proc. Int. Test Conf.*, pp. 932–938, 2001.

[Harris 2002] Harris, L.G. & Tessier, R., 2002. Testing and diagnosis of interconnect faults in cluster-based FPGA architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11), pp.1337–1343.

[Huang 1996] Huang, W. and Lombardi, F., 1996. An approach to testing programmable/ configurable field programmable gate arrays. In *Proc. 14th VLSI Test Symp*. pp. 450–455.

[Hughes 1986] J. L. A. Hughes and E. J. McCluskey, 1986. Multiple Stuck-at Fault Coverage of Single Stuck-at Fault Test Sets. In *Proc. Int. Test Conf*. pp. 368–374.

[ITRS 2013] ITRS, S.I., 2013. International technology roadmap for semiconductors: Executive summary. *Semiconductor Industry Association, Tech. Rep*..

[Kastensmidt 2006] Kastensmidt, Fernanda Lima, Reis, R., 2006. *Fault-Tolerance Techniques for SRAM-Based FPGAs*, Springer Publishers, US.

[Kim 2002] Yong Chang Kim; Agrawal, V.D.; Saluja, K.K., 2002. Multiple Faults : Modeling , Simulation and Test Design Automation Conference, 2002. *Proceedings of ASP-DAC, Asia and South Pacific and the 15th International Conference on VLSI Design*. Proceedings. Pp 592-597.

[Kumar 2013] Kumar, T.N. & Lombardi, F., 2013. A novel heuristic method for application-dependent testing of a SRAM-based FPGA interconnect. *IEEE Transactions on Computers*, 62(1), pp.163–172.

[Kuon 2007] Kuon, I., Tessier, R. & Rose, J., 2007. FPGA Architecture: Survey and Challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2), pp.135–253.

[Kuon 2008] Kuon, I. & Rose, J., 2008. Automated transistor sizing for FPGA architecture exploration. *Proceedings - Design Automation Conference*, pp.792–795.

[Kyria 2009] Kyriakoulakos, K. and Pnevmatikatos, D. A novel SRAM-based FPGA architecture for efficient TMR fault tolerance support. *In Field Programmable Logic and Applications, FPL 2009. International Conference on.*

[Legat 2010] Legat, U. Biasizzo, A. ; Novak, F. Automated SEU fault emulation using partial FPGA reconfiguration. Design and *Diagnostics of Electronic Circuits and Systems (DDECS), IEEE 13th International Symposium on,* pp 24-27, 2010

[Lemieux 2004] G. Lemieux and D. Lewis, 2004. *Design of interconnection networks for programmable logic*, Kluwer Academic Publishers.

[Lin 2010] Lin, M., Wawrzynek, J. & Gamal, A. El, 2010. Exploring FPGA routing architecture stochastically. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(10), pp.1509–1522.

[LIP6] http://www.lip6.fr/

[Marquardt 1999] A. S. Marquardt, V. Betz, and J. Rose, " Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density," in *Proc. Int. Symposium on Field Programmable Gate Arrays*, New York, USA. ACM, pp. 37-46, 1999

[Marrakchi 2010] Z. Marrakchi, H. Mrabet, and H. Mehrez, 2010. Programmable gate array, switch box and logic unit for such an array. Patent US 7795911, 2010

[Marrakchi 2009] Marrakchi, Z., Mrabet, H., Farooq, U. & Mehrez, H., 2009. FPGA Interconnect Topologies Exploration. *International Journal of Reconfigurable Computing*, 2009, pp.1–13.

[McCluskey 2004] McCluskey, E., Al-Yamani, A., Li, J., Tseng, C., Volkerin, E., Ferhani, F., Li, E., and Mitra, S., "ELF-Murphy data on defects and test sets," *Proc.22nd VLSI Test Symp., pp. 16–22, 2004.*

[McCracken 2002] S. McCracken and Z. Zilic, 2002. FPGA test time reduction through a novel interconnect testing scheme. In *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*. pp. 136–144.

[MCNC] "Lgsynth93 benchmark set: Version 4.0," Tech. Rep., 1993.

[Moore 1998] Moore, G.E., 1998. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1), pp.82–85.

[Naviner 2011] L. Naviner, J.-F. Naviner, G. dos Santos Jr., E. Marques, and N. P. Jr., "FIFA: A fault- injection-fault-analysis-based tool for reliability assessment at RTL level," Microelectronics Reliability, vol. 51, no. 911, pp. 1459 – 1463, 2011.

[Niamat 2005] Niamat, M.Y., Hejeebu, S.S. & Alam, M., 2005. A BIST approach for testing FPGAs using JBITS. *Proceedings - 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2005*, 2005, pp.267–268.

[ParisTech] http://www.telecom-paristech.fr/

[Pistorius 2003] J. Pistorius and M. Hutton, Placement Rent exponent calculation methods, Temporal behaviour and FPGA architecture evaluation, SLIP, 2003.

[Pervez 2011] P. Husain and H. Mehrez, *Application-specific mesh-based heterogeneous FPGA architectures*, Springer 2011.

[Renovell 1997] Renovell, M., Portal, J., Figueras, J., and Zorian, Y., "Test pattern and test configuration generation methodology for the logic of RAM-based FPGA," *Sixth Asian Test Symp.*, pp. 254–259, 1997.

[Renovell 1998] Renovell, M., Portal, J., Figueras, J., and Zorian, Y., "Testing the interconnect of RAM-based FPGAs," *IEEE Design and Test of Computers*, pp. 45–50, 1998

[RobustFPGA] https://robustfpga.wp.mines-telecom.fr/

[Rose 1990] J. Rose, R. J. Francis, D. Lewis, and P. Chow, "Architecture of field programmable gate arrays: The effect of logic block functionality on area efficiency," Proceedings of the IEEE, vol. 25, pp. 1217–1225, 1990.

[Rose 2012] Rose, Jonathan and Luu, Jason and Yu, Chi Wai and Densmore, Opal and Goeders, Jeffrey and Somerville, Andrew and Kent, Kenneth B. and Jamieson, Peter and Anderson, J., 2012. The VTR Project: Architecture and CAD for FPGAs from Verilog to Routing. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. pp. 77–86.

[RTAX] RTAX, Radiation tolerant FPGAs. *http://www.microsemi.com/products/fpga-soc/radtolerant-fpgas/rtax-s-sl*.

[S. May 2006] May, G.S. and C.J.S., 2006. *Fundamentals of Semiconductor Manufacturing and Process Control*, John Wiley & Sons, Inc., Hoboken, New Jersey

[Seifert 2006] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill, and J. Maiz, "Radiation-induced soft error rates of advanced cmos bulk devices," in *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, 2006, pp. 217– 225.

[Smith 2006] J. Smith, T.X. and C.S., 2006. An Automated BIST Architecture for Testing and Diagnosing FPGA Interconnect. *Journal of electronic testing*, 22, pp.239–253.

[Stroud 1998] Stroud, C., Wijesuriya, S., Hamilton, C. & Abramovici, M., 1998. Built-in self-test of FPGA interconnect. *IEEE International Test Conference (ITC)*, pp.404–411.

[Stroud 2002] Stroud, C.E., 2002. *A Designer's Guide To BIST*, Kluwer Academic Pulishers.

[Sundararajan 2001] Sundararajan, P., Mcmillan, S., Guccione, S.A., Jose, S., et al., 2001. Testing FPGA Devices Using JBits. , 95124.

[Tahoori 2003] Tahoori, M. and Mitra, S., "Automatic configuration generation for FPGA interconnect testing," *Proc. 21st VLSI Test Symp*, pp. 134–139, 2003

[TIMA] http://tima.imag.fr/

[VPRTO] VPR: Package, http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html

[Wang 1998] Wang, S. and Huang, C., "Testing and diagnosis of interconnect structures in FPGAs," *Proc. Seventh Asian Test Symp.*, pp. 283–287, 1998.

[Wang 2007] Yabin Wang; Yuan Wang; Jinmei Lai , An FPGA configuration circuit used for fast and partial configuration, ASIC, 2007. ASICON '07. 7th International Conference on. pp, 157-160, 2007

[Xilinx] Xilinx, 2013. ISE Design Suite. *http://www.xilinx.com/products/design-tools/ise-design-suite.html*.

[XilinxRpt] Xilinx, June 2015, 7series FPGAs configuration user guide. http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[Yao 2009] Yao, J., Dixon, B., Stroud, C. & Nelson, V., 2009. System-level built-in self-test of global routing resources in virtex-4 FPGAs. *2009 IEEE International Symposium on Sustainable Systems and Technology, ISSST 2009*, pp.29–33.

[Yu 2005] Yu, A.J.; Lemieux, G.G.F. , Defect-tolerant FPGA switch block and connection block with fine-grain redundancy for yield enhancement, *Field Programmable Logic and Applications, 2005. International Conference on*, pp, 255 - 262

[Zhu 2011] Zhu, J., Hu, H., Dong, W. & Pan, L., 2011. A cost-efficient self-configurable BIST technique for testing multiplexer-based FPGA interconnect. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 27(5), pp.647–655.

TITRE

Développement des techniques de test et de diagnostic pour les FPGAs hiérarchique de type mesh.

RESUME

L'évolution tendant à réduire la taille et augmenter la complexité des circuits électroniques modernes, est en train de ralentir du fait des limitations technologiques, qui génèrent beaucoup de d'imperfections et de defaults durant la fabrication ou la durée de vie de la puce. Les FPGAs sont utilisés dans les systèmes numériques complexes, essentiellement parce qu'ils sont reconfigurables et rapide à commercialiser. Pour garder une grande fiabilité de tels systèmes, les FPGAs doivent être testés minutieusement pour les defaults. L'optimisation de l'architecture des FPGAs pour l'économie de surface et une meilleure routabilité est un processus continue qui impacte directement la testabilité globale et de ce fait, la fiabilité. Cette thèse présente une stratégie complète pour le test et le diagnostique des defaults de fabrication des "mesh-based FPGA" contenant une nouvelle topologie d'interconnections à plusieurs niveaux, ce qui promet d'apporter une meilleure routabilité. Efficacité des schémas proposes est analysée en termes de temps de test, couverture de faute et résolution de diagnostique.

MOTS CLEFS

Built-In Self-Test, FPGA hiérarchique de type mesh, Multilevel interconnect, Off-line test et diagnostic, Logic et interconnect BIST

TITLE

Development of test and diagnosis techniques for hierarchical mesh-based FPGAs

ABSTRACT

The evolution trend of shrinking feature size and increasing complexity in modern electronics is being slowed down due to physical limits that generate numerous imperfections and defects during fabrication steps or projected life time of the chip. Field Programmable Gate Arrays (FPGAs) are used in complex digital systems mainly due to their reconfigurability and shorter time-to-market. To maintain a high reliability of such systems, FPGAs should be tested thoroughly for defects. FPGA architecture optimization for area saving and better signal routability is an ongoing process which directly impacts the overall FPGA testability, hence the reliability. This thesis presents a complete strategy for test and diagnosis of manufacturing defects in mesh-based FPGAs containing a novel multilevel interconnects topology which promises to provide better area and routability. Efficiency of the proposed test schemes is analyzed in terms of test cost, respective fault coverage and diagnostic resolution.

KEYWORDS

Built-In Self-Test, Hierarchical mesh of clusters FPGA, Multilevel interconnect, Off-line test and diagnosis, Logic and interconnect BIST

INTITULE ET ADRESSE DU LABORATOIRE

Laboratoire TIMA, 46 Avenue Félix Viallet, 38031 Grenoble, France.